Evaluating Predicates over Encrypted Data

Elaine Shi

CMU-CS-08-166

October, 2008

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Adrian Perrig (CMU), Chair
Dawn Song (CMU/U.C.Berkeley)
Manuel Blum (CMU)
Brent Waters (SRI/U.T.Austin)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was supported in part by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389, grants CNS-0347807 and CT-CS 0433540 from the National Science Foundation, a grant from General Motors through the GM-CMU Collaborative Research Laboratory, and by a gift from Bosch.

The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, Bosch, CMU, GM, NSF, or the U.S. Government or any of its agencies.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
OCT 2008		**00-00-2008 to 00-00-2008**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Evaluating Predicates over Encrypted Data	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,School of Computer Science,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**137**	

Keywords: Predicate encryption, applied cryptography, bilinear group

Abstract

Predicate encryption is a new encryption paradigm where the secret key owner can perform fine-grained access control over the encrypted data. In particular, the secret key owner can generate a capability corresponding to a query predicate (e.g., whether an encrypted email contains the keyword MEDICAL), and the capability allows one to evaluate the outcome of this predicate on the encrypted data.

The high-level goal of this thesis is to build predicate encryption systems that are efficient, support expressive queries and rich operations. Our contributions are summarized below:

1. We propose a predicate encryption scheme supporting *multi-dimensional range queries*. Prior to this work, researchers have constructed schemes support equality tests. Hence, our scheme supports more expressive queries than before. At the core of this construction is a technique to support conjunctive queries without leaking the outcome of each individual clause.
2. We study how to *delegate capabilities* in predicate encryption schemes. To demonstrate why delegation may be interesting, imagine that Alice has a capability, and she wishes to delegate to Bob a more restrictive capability allowing him to decrypt a subset of the information Alice can learn about the plaintext encrypted. We propose a security definition for delegation, and build a scheme supporting delegation and conjunctive queries.
3. Most prior work focuses on hiding the plaintext (encoded in the ciphertext), but does not provide guarantees about the secrecy of the queries (encoded in the capabilities). In other words, given a capability, one might be able to infer from it what the query predicate is. We study how to *hide the query predicates*, and propose a scheme supporting inner-product queries that hides the query predicates in addition to the plaintext.

Acknowledgments

I am indebted to my thesis committee:

- the “big boss” Adrian,
- my “pseudo-advisor” Dawn,
- my “academic uncle” Brent,
- and the most kind and understanding Manuel.

Without your help, this thesis would not have been possible.

Apart from my committee members, John Bethencourt, Hubert Chan and Emily Shen also contributed to this thesis.

I would also like to thank the English and Greek alphabet, for I am sure the notations would have become more confusing, had I used Chinese characters to denote things.

Contents

Chapter 1

Introduction

1.1 What is Predicate Encryption?

Alice loves Gmail. However, she is concerned about her privacy, and she does not wish Google to read her emails. A common approach to address such privacy concerns is to use encryption. Imagine that Alice now uses traditional public-key encryption to protect the secrecy of her emails. Alice generates a public-key/private-key pair. She publishes the public-key PK, so her friends can encrypt the emails using PK before sending them to Alice. Now all emails will be stored in encrypted format at Google, and Alice is happy about being able to protect her privacy.

Now Alice wishes to search for all emails whose ”(sender = Bob) and (date within [2006, 2007])”. Unfortunately, Google can no longer search her emails, since the emails are stored in encrypted format, and without the secret key, the emails are indistinguishable from random numbers to Google. Alice can download all emails from Google, decrypt and search them locally. But what if there are too many emails to download? Alternatively, Alice can give away her private-key to Google, but of course, that beats the purpose of encryption.

This problem can be solved using a new type of encryption, called *predicate encryption*. Using predicate encryption, Alice can compute a capability corresponding to her query, e.g., ”(sender = Bob) and (date within [2006, 2007])”. She gives this capability to Google, and Google can test the capability against Alice’s encrypted emails. In this way, Google is able to learn which emails match the query; and beyond this information, Google learns nothing more about the encrypted emails. In contrast to traditional encryption, predicate encryption offers the property that access to the plaintext is no longer all-or-nothing. One can release partial information about the encrypted data in a controlled manner.

1.2 Related Work

In traditional public key encryption a user creates a public and private key pair where the private key is used to decrypt all messages encrypted under that public key. While this functionality is sufficient for applications where a one-to-one association exists between a particular user and a public key, several applications will demand a finer-grained and more expressive decryption

capabilities. Shamir [33] was the first to introduce finer-grained encryption systems by defining the concept of Identity-Based Encryption (IBE). In an IBE system a party encrypts a message under a particular public key and associates the ciphertext with a given string or "identity". A user can obtain a private key, that is derived from a master secret key, for a particular identity and can use it to decrypt any ciphertext that was encrypted under his identity.

Since the realization of the first Identity-Based Encryption schemes by Boneh and Franklin [7] and Cocks [18], there have been a number of new cryptosystems that provided increasing functionality and expressiveness of decryption capabilities. In Attribute-Based Encryption systems (ABE) [2, 16, 26, 31, 32] a user can receive a private capability that represents a complex access control policy over the attributes of an encrypted record. Other encryption systems, including keyword search (or anonymous IBE) [1, 6, 9, 12, 13, 22, 28, 34, 35, 36] systems, allow for a capability holder to evaluate a predicate on the the encrypted data itself and learn nothing more. This type of functionality represents a significant breakthrough in the sense that access to the encrypted data is no longer all-or-nothing; a user with a predicate capability will be able to learn partial information about encrypted data.

1.3 Applications of predicate encryption

Apart from the private Gmail scenario described above, predicate encryption also has various other applications.

Network audit logs. Recently, the network intrusion detection community has made large-scale efforts to collect network audit logs from different sites [20, 21, 30]. In this application, a network gateway or an Internet Service Provider (ISP) can submit network traces to an audit log repository. However, due to the presence of privacy sensitive information in the network traces, the gateway will allow only authorized parties to search their audit logs. We consider the following four types of entities: a *gateway*, an *untrusted repository*, an *authority*, and an *auditor*. Predicate encryption allows the gateway to submit encrypted audit logs to the untrusted repository. Normally, no one is able to decrypt these audit logs. However, when malicious behavior is suspected, an auditor may ask the authority for a search capability. With this search capability, the auditor can decrypt entries satisfying certain attack characteristics, e.g., network flows whose destination address and port number fall within a certain range. However, the privacy of all other flows should still be preserved. Note that in practice, to avoid a central point of trust, we can have multiple parties to jointly act as the authority. Only when a sufficient number of the parities collaborate, can they generate a valid search capability. Securely splitting the authority into multiple parties can be achieved through secure multi-party computation techniques [24], and is outside the scope of this thesis.

Financial audit logs. Financial audit logs contain sensitive information about financial transactions. Predicate encryption allows financial institutions to release audit logs in encrypted format. When necessary, an authorized auditor can obtain a decryption key from a trusted authority. With

this decryption key, the auditor can decrypt certain transactions that may be suspected of fraudulent activities. However, the privacy of all other transactions is preserved.

Public health monitoring. Consider a health monitoring program. When Alice moves about in her daily life, a PDA or smart-phone she carries automatically deposits her encrypted location at a storage server. Assume that each crumb is of the form $((x, y, t), ct)$, where (x, y) represents the location, t represents time, and ct is Alice's contact information. During an outbreak of an epidemic, Alice wishes to be alerted if she was present at a site borne with the disease during an incubation period, i.e., if (x, y, t) falls within a certain range. However, she is also concerned with privacy, and she does not wish to leak her trajectory if she has not been to a site borne with the disease.

Sharing of medical records. Medical research institutes would like to obtain patients' medical records for their research. However, these medical records are usually privacy sensitive, and it is necessary to enforce access control, such that a cardiologist is allowed to access medical records related to heart diseases, but not records on eye diseases. Using predicate encryption, we can easily enforce such fine-grained access control policies by granting the cardiologist a capability that allows her to decrypt precisely the medical records she needs. Furthermore, if the cardiologist would like her assistant to check all cases that happen within the year 2008, she can perform a delegation operation, and generate a sub-capability that allows the decryption of all records on heart diseases and within the year 2008.

Stock trading through an untrusted broker. An investor uses a broker to trade stocks. The investor does not fully trust the broker, and wishes to reveal as little information to the broker as possible. For example, the investor can place an order that says, "buy x amount of stock y if the price falls below p today". The broker should not be able to decrypt this order until the current price satisfies the conditions specified by the order. This problem can be addressed through predicate encryption. A party trusted by the investor (e.g., the stock exchange) issues a new capability to the broker as the stock price changes. The broker can now try to use the capability to decrypt the investor's order. If the current price meets the conditions specified by the order, the decryption is successful, and the order gets executed. If the order is never executed, the broker learns nothing about the contents of the order, except the fact that the conditions specified by the order were never met.

Untrusted remote storage. Individual users may wish to store emails and files on a remote server, but because the storage server is untrusted, the content must be encrypted before it is stored at the remote server. Emails and files can be classified with multiple attributes. Users may wish to perform certain types of queries and retrieve only data that satisfy the queries.

Using biometrics in anonymous IBE. Predicate encryption can also be used in biometric-based Anonymous Identity-Based Encryption (AIBE). Using biometrics in identity-based encryption first appeared in the work by Sahai and Waters [32]. In this application, a person's biometric features

such as finger-prints, blood-type, year of birth, eye color, etc., are encoded as a point $\mathbf{X}$ in a multi-dimensional lattice. Personal data is encrypted using the owner's biometric features as the identity, and the encryption protects both the secrecy of the personal data and the owner's biometric identity. Due to potential noise each time a person's biometric features are sampled, a user holding the private key for biometric identity $\mathbf{X}$ should be allowed to decrypt data encrypted under $\mathbf{X}'$, *iff* $\mathbf{X}'$ and $\mathbf{X}$ have small distance. In particular, the SahaiWaters04 construction [32] considered the *set-overlap* distance (or the *Hamming* distance); and their encryption scheme does not hide the identity of the user. Our construction for multi-dimensional queries allows a user with the private key for identity $\mathbf{X}$, to decrypt an entry encrypted under $\mathbf{X}'$, iff $\ell_\infty(\mathbf{X}, \mathbf{X}') \leq \epsilon$. Here ℓ_∞ denotes the ℓ_∞ distance between $\mathbf{X}$ and $\mathbf{X}'$, and is defined as $\max\{|x_1 - x'_1|, \ldots, |x_D - x'_D|\}$. In this case, the decryption region is a hyper-cube in multi-dimensional space. One can also associate a different weight to each dimension, in which case the decryption region becomes a hyper-rectangle.

1.4 Efficiency and expressiveness

One important goal in designing predicate encryption systems is the ability to support complex query predicates. Meanwhile, we would like our construction to be efficient. To be specific about what we mean by efficient, we consider the following performance metrics: encryption time, ciphertext size, capability size and decryption time. Ideally, we would like all of these performance metrics to be polynomial in the length of the plaintext (and also the security parameter).

Previously, researchers have designed predicate encryption schemes that support keyword-based searches [1, 6, 9, 39]. A keyword search is an equality test: given a specially formed capability, one can evaluate whether the ciphertext is an encryption of a specific plaintext. For example, if we use such a predicate encryption system for the above-mentioned network audit log application, the auditor would be able to make queries of the form: "PORT$= 1434$". (This is the typical port number used by the SQL Slammer worm.)

1.5 Summary of contributions

The high-level goal of this thesis is to develop predicate encryption schemes that (1) are efficient, (2) support expressive queries and rich operations, and (3) have better security (under certain assumptions). The main technical content of this thesis is formed around three papers [34, 35, 36], each of which proposes a novel construction, and represents an endeavor at the above-mentioned high-level goal. Table 1.1 summarizes the contributions of each of these papers. The work on multi-dimensional range query is done with J. Bethencourt, H. Chan, D. Song and A. Perrig; the work on delegation of capabilities is joint with Brent Waters; and the work on query-hiding predicate encryption is joint with Emily Shen and Brent Waters.

To help readers understand the development of this field, position our work, and understand our contributions, I also created Table 1.2 which lists related work in the area of predicate encryption.

Constructions	Expressiveness and features	Comments	Contribution
SBCSP07 [35]	*Multi-dimensional range query*	Secure in the match-revealing model	More expressive queries
SW08 [36]	Conjunctive query, *Delegation*		Richer operations (delegation)
SSW08 [34]	Inner-product query, *Hides query* in addition to plaintext	Secret-key setting	Stronger security, more expressive

Table 1.1: **Summary of contributions**. See chapter 2 for the definition of match-revealing security, secret-key setting, etc.

Constructions	Expressiveness and features	Example query	Comments
[1, 6, 9, 13, 22, 39]	Equality test query	SENDER = Bob	a.k.a. Keyword searches
[12, 25]	Conjunctive queries & extensions	(SENDER = Bob) $\wedge$ (YEAR = 2008)	[25] reveals the outcome of individual clauses
SBCSP06 [35]	*Multi-dimensional range query*	(URGENT $\in$ [0, 3]) $\wedge$ (YEAR $\in$ [2003, 2008])	Secure in the match-revealing model
SW08 [36]	Conjunctive query, *Delegation*	(SENDER = Bob) $\wedge$ (YEAR = 2008)	
[28]	Inner-product query	$(\vec{x}, \vec{v}) = 0$	
SSW08 [34]	Inner-product query, *Hides query* in addition to plaintext	$(\vec{x}, \vec{v}) = 0$	Secret-key setting

Table 1.2: **Putting it in context with related work**. This table summarizes our work in this space and positions our work in context with related work. The highlighted constructions are the contributions of this thesis. The table is created roughly (not strictly) in chronological order. The BW06 construction [12] and the SBCSP06 [35] construction are independent and concurrent work. It is difficult to construct an intuitive query example for inner-product queries. However, note that inner-product is strictly more expressive than conjunctions. See Chapter 5 for more details on inner product queries.

Outline of the thesis. Chapter 2 presents a formal definition of predicate encryption and its security. Chapter 3, 4 and 5 will each describe the one of the results shown above in Table 1.1. While these three chapters are logically connected, each chapter is self-contained and can be read independently from others.

A note on the notations. Chapter 2 gives a generic and unified definition. As the following Chapters 3, 4 and 5 each considers a more specific scenario, we will use a more concrete instantiation of the generic definition in each chapter. The resulting variation in definition and notation across chapters is intended for notational convenience, and does not affect the essence of generic definition given in Chapter 2. These variants of definitions and notations will be clearly stated in each chapter to avoid ambiguity.

Chapter 2

Formal Definitions

Recall that in predicate encryption, a party who owns the master secret key can generate a capability (also referred to as a *token*) that allows one to decrypt all data entries satisfying a certain predicate function f. However, all other information about the plaintext still remains secret.

In this chapter, we give formal definitions for predicate encryption and its security.

2.1 Public-key predicate encryption

We now give a formal definition for public-key predicate encryption. This definition is due to Boneh and Waters [12].

Let $X = (x_1, x_2, \ldots, x_\ell) \in \{0,1\}^\ell$ denote a plaintext. Without loss of generality, assume that we would like to evaluate from the ciphertext boolean functions (also referred to as *predicates*) on X, that is $f : \{0,1\}^\ell \to \{0,1\}$. Functions that output multiple bits can be regarded as concatenation of boolean functions. Let $\mathcal{F}$ denote a family of boolean functions from $\{0,1\}^\ell$ to $\{0,1\}$. For example, $\mathcal{F}$ can be the set of all conjunctions on $(x_1, x_2, \ldots, x_\ell) \in \{0,1\}^\ell$. A token allows one to evalute from the ciphertext a predicate $f \in \mathcal{F}$.

A Public-Key Predicate Encryption (PK-PE) scheme consists of the following (possibly randomized) algorithms.

$Setup(1^\lambda, \mathcal{F})$**.** The $Setup$ algorithm takes as input a security parameter 1^λ the predicate family $\mathcal{F}$ being considered; and outputs a public key PK and a master secret key MSK.

$Encrypt(\mathsf{PK}, X)$**.** The $Encrypt$ algorithm takes as input a public key PK, a plaintext $X = (x_1, x_2, \ldots, x_\ell) \in \{0,1\}^\ell$; and outputs a ciphertext CT.

$GenToken(\mathsf{PK}, \mathsf{MSK}, f)$**.** The $GenToken$ algorithm takes as input a public key PK, master secret key MSK, and a query predicate $f \in \mathcal{F}$. It outputs a token for evaluating the predicate f from a ciphertext.

$Query(\mathsf{PK}, \mathsf{CT}, \mathsf{TK}_f)$**.** The $Query$ algorithm takes as input a public key PK, a token TK_f for the predicate f, and a ciphertext CT. Suppose CT is an encryption of the plaintext X; the algorithm outputs $f(X)$.

2.1.1 Security definitions

To define the security for predicate encryption, we describe a query security game between a challenger and an adversary. This game formally captures the notion that the tokens reveal no unintended information about the plaintext. In this game, the adversary asks the challenger for a number of tokens. The adversary should not be able to deduce any unintended information from these tokens. The game proceeds as follows:

- **Setup.** The challenger runs the $Setup$ algorithm, and gives the adversary the public key PK.
- **Query 1.** The adversary adaptively makes a polynomial number of queries. In each query, the adversary specifies a predicate $f \in \mathcal{F}$, and asks the challenger for a token for that predicate. The challenger computes the requested token by calling the $GenToken$ algorithm, and returns the token to the adversary.
- **Challenge.** The adversary outputs two strings $X_0^*, X_1^* \in \{0,1\}^\ell$ subject to the constraint that for any predicate f queried by the adversary in the **Query 1** stage, the following must be true:

  $$f(X_0^*) = f(X_1^*) \tag{2.1}$$

  Next, the challenger flips a random coin b, and encrypts X_b^*. It returns the ciphertext to the adversary.
- **Query 2.** Repeat the **Query 1** stage. All predicates queried in this stage should satisfy the same condition as above.
- **Guess.** The adversary outputs a guess b' of b.

The advantage of an adversary $\mathcal{A}$ in the above game is defined to be $\mathsf{Adv}_{\mathcal{A}} = |\Pr[b = b'] - 1/2|$.

Definition 2.1.1 *We say that a public-key predicate encryption system is* secure*, if for all polynomial time adversaries $\mathcal{A}$ attacking the system, its advantage $\mathsf{Adv}_{\mathcal{A}}$ is a negligible function of λ.*

2.1.2 Selective security

We also define a weaker security notion called *selective security*. In the selective security game, instead of submitting two strings X_0^*, X_1^* in the **Challenge** stage, the adversary first commits to two strings at the beginning of the security game. The rest of the security game proceeds exactly as before. The selective security model has appeared in various constructions in the literature [3, 12, 13, 14, 15, 35], since it is often easier to prove security in the selective model.

Definition 2.1.2 *We say that a delegateable predicate encryption system is* selectively secure*, if all polynomial time adversaries $\mathcal{A}$ have negligible advantage in the selective security game.*

2.1.3 Match revealing security

In a recently published paper [35], we define another relaxed version of security called *match-revealing* security. In comparison, we call the strict version of security (as defined in Section 2.1.1) *match-concealing* security.

In *match-concealing* security, the adversary does not learn any additional information about the plaintext whether or not the output of the predicate is true. The readers can think of this as "two-sided" security. By contrast, *match-revealing* security can be thought of as "one-sided" security:

- When the predicate evaluates to true, the adversary does not learn any additional information about the plaintext encrypted;
- When the predicate evaluates to false, we no longer care about preserving the secrecy of the plaintext.

Clearly match-concealing security implies match-revealing security. However, we are also interested in match-revealing security, because in some cases, using the relaxed version of security can lead to more efficient and practical constructions. Meanwhile, in many practical applications, we no longer care about the secrecy of the encrypted entry if it matches the query predicate. For example, in the above-mentioned network audit log example, a matching entry corresponds to a suspicious or attack flow. In this case, the audit is interested in decrypting the entire entry and studying it. Hence, we are not obligated to preserve the privacy of these matching entries. Of course, one can also conceive of other applications where the strict notion of security, that is, match-concealing security, is necessary.

The formal definition of match-revealing security is almost the same as match-concealing security, with the exception that Equation (2.1) is now the following new equation:

$$f(X_0^*) = f(X_1^*) = 0$$

2.2 Secret-key predicate encryption

Secret-key predicate encryption can be similarly defined as public-key predicate encryption. The difference is that in public-key encryption, anyone can encrypt using the public-key. By contrast, in secret-key encryption, encryption and decryption are both performed using the secret-key. Hence, only the secret-key owner can encrypt. In both schemes, only the secret-key owner can decrypt.

We now define secret-key predicate encryption. More discussion on the security definitions can be found in Chapter 5.

A Secret-Key Predicate Encryption (SK-PE) scheme consists of the following (possibly randomized) algorithms.

Definition 2.2.1 (Secret-key predicate encryption) *A Secret-Key Predicate Encryption (SKPE) system consists of the following (possibly randomized) algorithms.*

$Setup(1^\lambda)$: The $Setup$ algorithm takes as input a security parameter 1^λ, and outputs a secret key MSK.

$Encrypt(\mathsf{MSK}, x)$: The $Encrypt$ algorithm takes as input a secret key MSK, a plaintext $x \in \{0,1\}^\ell$; and outputs a ciphertext CT.

$GenToken(\mathsf{MSK}, f)$: The $GenToken$ algorithm takes as input a secret key MSK, and a query predicate $f : \{0,1\}^\ell \to \{0,1\}$. It outputs a token TK_f that allows one to evaluate $f(x)$ over an encryption of x. As mentioned above, we assume that the query predicate can be encoded with a bitstring of length m.

$Query(\mathsf{TK}_f, \mathsf{CT})$: The $Query$ algorithm takes as input a token TK_f for the predicate f, and a ciphertext CT which is an encryption of $x \in \{0,1\}^\ell$, the algorithm outputs $f(x)$.

2.2.1 Security definitions: Hiding both the plaintext and the query

Public-key predicate encryption schemes guarantee the secrecy of the ciphertext; however, they do not guarantee the secrecy of the tokens. In fact, for public-key predicate encryption, it is inherently impossible to achieve ciphertext secrecy and token secrecy simultaneously. This is due to the fact that anyone is able to encrypt using the public-key. In the Gmail example, if Google would like to know whether a token corresponds to the query "TITLE = cryptography", Google can simply encrypt an email whose "TITLE = cryptography" using the public-key, and test the token against the resulting ciphertext.

In secret-key predicate encryption, it is possible to guarantee the secrecy of both the plaintext (encoded in a ciphertext) and that of the query (encoded in a token). This provides even stronger privacy guarantees in practice.

We now formally define the security for secret-key predicate encryption. As mentioned above, our definition aims to guarantee the secrecy of the plaintext, as well as the query.

To explain the intuition behind our security definition, consider a privacy-preserving remote storage application, where Alice stores her encrypted documents on a remote server, and later issues tokens to the server to search for matching documents. Our goal is to leak as little information to the storage server as possible. Under our model, Alice makes a query by submitting a token to the server, and the server learns exactly which of her encrypted documents match the query, and returns the matching documents to Alice. Therefore, in this framework, the server inevitably learns Alice's *access pattern*, a.k.a, which documents Alice retrieves with each query.

We would like to define security in the strongest sense possible: informally, *the storage server should learn only Alice's access pattern, and nothing more*. In particular, this implies that the server learns nothing about Alice's encrypted documents, or what queries she is making.

To capture the notion that the server learns only Alice's access pattern, we need to first formally define what *access pattern* means. Intuitively, the access pattern is the outcomes of q predicates on n plaintexts.

Definition 2.2.2 (Access pattern) *Let $X = (x_1, x_2, \ldots, x_n)$ denote an ordered list of n plaintexts, where $x_i \in \{0,1\}^\ell$ for $1 \le i \le n$. Let $F = (f_1, f_2, \ldots, f_q)$ denote an ordered list of q query predicates, where $f_i \in \{0,1\}^m$ for $1 \le i \le q$. The access pattern on X and F is an $q \times n$ matrix:*

$$\textsc{AccessPattern}(X, F) := \begin{bmatrix} f_1(x_1), & f_1(x_2), & \ldots, & f_1(x_n) \\ f_2(x_1), & f_2(x_2), & \ldots, & f_2(x_n) \\ \ldots, \ldots & & & \\ f_q(x_1), & f_q(x_2), & \ldots, & f_q(x_n) \end{bmatrix}$$

We now proceed to define the security for SKPE. Let

$$X = (x_1, x_2, \ldots, x_n), \quad X' = (x'_1, x'_2, \ldots, x'_n)$$

denote two ordered lists of plaintexts. Let

$$F = (f_1, f_2, \ldots, f_q), \quad F' = (f'_1, f'_2, \ldots, f'_q)$$

denote two ordered lists of queries predicates. Now imagine the following two worlds. In World 0, the server sees n encrypted documents and q tokens:

$$\left(\mathsf{Enc}(x_1), \mathsf{Enc}(x_2), \ldots, \mathsf{Enc}(x_n)\right), \quad \left(\mathsf{TK}_{f_1}, \mathsf{TK}_{f_2}, \ldots, \mathsf{TK}_{f_q}\right)$$

In World 1, the server sees n encrypted documents and q tokens:

$$\left(\mathsf{Enc}(x'_1), \mathsf{Enc}(x'_2), \ldots, \mathsf{Enc}(x'_n)\right), \quad \left(\mathsf{TK}_{f'_1}, \mathsf{TK}_{f'_2}, \ldots, \mathsf{TK}_{f'_q}\right)$$

Suppose the two worlds have the same access pattern, i.e.,

$$\textsc{AccessPattern}(X, F) = \textsc{AccessPattern}(X', F')$$

Informally, the server should not be able to distinguish between the two worlds. The security definition presented below describes a game between a challenger and an adversary, and is intended to capture this notion of indistinguishability between two these worlds. Moreover, the definition considers an adaptive adversary: an adversary who can choose what ciphertext/token queries to make depending on the previous interactions with the challenger.

Definition 2.2.3 (SKPE full security) *We say that an SKPE scheme is fully secure, if no polynomial-time adversaries has more than negligible advantage in the following game.*

Setup. The challenger runs the $Setup$ algorithm, and retains the secret key MSK to itself. In addition, it flips a random coin b, and keeps the bit b to itself as well. Define four ordered lists, X_0, F_0, X_1, F_1, where (X_0, F_0) will record plaintexts and predicates queried by the adversary in World 0, and (X_1, F_1) will record plaintexts and predicates queried by the adversary in World 1. Initially, all four lists are empty.

Query. The adversary adaptively makes the following types of queries. The adversary can make up to a polynomial number of these queries.

- **Ciphertext query.** The adversary specifies two plaintexts $x_0, x_1 \in \{0,1\}^\ell$ to the challenger. The challenger encrypts x_b and returns the ciphertext to the adversary. Append x_0 to the list X_0, and x_1 to the list X_1.
- **Token query.** The adversary specifies two predicates $f_0, f_1 \in \{0,1\}^m$ to the challenger. The challenger computes a token for the predicate f_b, and gives the resulting token to the adversary. Append f_0 to the list F_0, and f_1 to the list F_1.

All queries made in this stage should be indistinguishable by access pattern. In other words, at the end of the game, all queries made should satisfy the following condition:

$$\textsc{AccessPattern}(X_0, F_0) = \textsc{AccessPattern}(X_1, F_1)$$

Guess. The adversary outputs a guess b' of the bit b. Its advantage is defined as $\mathbf{Adv}_\mathcal{A} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

Chapter 3

Multi-Dimensional Range Query over Encrypted Data

3.1 Multi-dimensional Range Queries over Encrypted Data

In this section, we demonstrate a predicate encryption system supporting multi-dimensional range queries. This contents of this section are based on work published in a recent paper [35]. Therefore, throughout this section, we use notations consistent with the paper [35].

The reason that we are particularly interested in multi-dimensional range queries is because they are the most prevalent type of queries in current database systems. In fact, SQL queries are by nature multi-dimensional range queries.

3.1.1 Overview of our construction

We assume that each plaintext entries has D attributes, and the query predicates are conjunctions of range queries over a subset of these D attributes. For example, assume that each entry has the structure (IP, port, time), and below is a typical example of a multi-dimensional range query:

$$(\text{IP} \in [128.2.*.*]) \wedge (\text{time} \in \{2006, 2007\})$$

A more formal and complete definition will be given in Section 3.1.2.

We give a provably secure predicate encryption system for multi-dimensional range queries. The performance of our construction can be summarized by Table 3.1.

Comparison with BonehWaters06. In the above table, the BonehWaters06 scheme is concurrent and independent work to ours. In their paper, they give a general definition for predicate encryption, and propose a scheme called Hidden Vector Encryption (HVE) for performing conjunctive equality tests. They then show how HVE can be extended to support conjunctive subset/range queries.

The HVE construction given by Boneh and Waters has ciphertext length and encryption time linear with respect to the length of the plaintext. The token size has length linear in the number

Scheme	Pub. Key Size	Encrypt. Cost	CT Size	Token Size	Decrypt. Cost	Security
BonehWaters06 [12]	$O(D \cdot T)$	$O(D \cdot T)$	$O(D \cdot T)$	$O(D)$	$O(D)$	MC
Naive AIBE-based	$O(1)$	$O((\log T)^D)$	$O((\log T)^D)$	$O((\log T)^D)$	$O((\log T)^D)$	MR
Our scheme	$O(D \cdot \log T)$	$O(D \cdot \log T)$	$O(D \cdot \log T)$	$O(D \cdot \log T)$	$O((\log T)^D)$	MR

Table 3.1: Performance of different approaches. D denotes the number of dimensions and T the number of points in each dimension. The naive AIBE-based scheme is described in Section 3.2.3. MC and MR refer to the *match-concealing* and *match-revealing* security models respectively.

of clauses in the conjunctive query, and so is decryption time. The HVE construction is efficient in the sense that all performance measures are polynomial (in fact, linear) in the length of the plaintext encrypted. However, to extend HVE to support multi-dimensional range queries, they need to incur an exponential cost. Notice that in Table 3.1, both public key, ciphertext size and encryption time are $O(D \cdot T)$, and this is exponential in the length of T (i.e., number of bits needed to encode T distinct values).

Our construction [35] is very similar to the BonehWater06 work in many ways. We also use pairing based techniques to build our construction. Although the two schemes appear different at the algebra level, (for example, the two constructions use different types of bilinear groups) at the core of both constructions is a similar idea to defend against the collusion attack (See Section 3.3.2). In particular, although not explicitly stated, the core of our construction is also an HVE-like scheme that supports a conjunction of equality tests, and it can be proven secure in the match-concealing security model. However, we encountered similar difficulties when we endeavored to extend it to support multi-dimensional range queries, essentially, we had to incur a significant cost which would have made the construction too expensive in practice, and typically, in the network audit log and similar applications.

To cope with such difficulties, we propose the relaxed security notion, that is, match-revealing security. Our multi-dimensional range query construction uses the relaxed security notion instead. Doing this allows us to enable a better trade-off in the various performance measures. As shown by Table 3.1, our construction has $O(D \log T)$ public key size, ciphertext size and encryption time. In comparison, the BonehWaters06 construction has $O(DT)$ public key size, ciphertext size and encryption time. On the other hand, our construction is more expensive in decryption time. We need $O((\log T)^D)$ decryption time while the BonehWater06 construction has a decryption time of $O(D)$. In applications like network audit logs as described above, T can be as large as 2^{32} to encode an IP address, and typically, D may range from 2 to 4. In such scenarios where T is large and D is small, our construction is more practical. However, one can also conceive of other applications where T is small and D is large, and in these cases, the BonehWaters06 construction would be more practical.

3.1.2 Definitions

In the network audit log application, a gateway encrypts network flows, and submits them to an untrusted repository. When necessary, an auditor may ask an authority for a key that allows the decryption of all flows whose attributes fall within a certain range; while the privacy of all irrelevant

flows are still preserved. There is a geometric interpretation to these multi-attribute range queries. Suppose that we would like to allow queries on these three fields: time-stamp t, source address a, and destination port p. The tuple (t, a, p) can be regarded as a point $\mathbf{X}$ in multi-dimensional space. Now suppose we query for all flows whose t, a, p falls within some range: $t \in [t_1, t_2]$, $a \in [a_1, a_2]$ and $p \in [p_1, p_2]$. Here the "hyper-range" $[t_1, t_2] \times [a_1, a_2] \times [p_1, p_2]$ forms a hyper-rectangle $\mathbf{B}$ in space. The above range query is equivalent to testing whether a point $\mathbf{X}$ falls inside the hyper-rectangle $\mathbf{B}$.

We now formally define these notions mentioned above. Assume that an attribute can be encoded using discrete integer values 1 through T. For example, an IP address can be encoded using integers 1 through 2^{32}. We use the notation $[T]$ to denote integers from 1 to T, i.e., $[T] = \{1, 2, \ldots, T\}$. Let $S \leq T$ be integers, we use $[S, T]$ to denote integers from S to T inclusive, i.e., $[S, T] = \{S, S+1, \ldots, T\}$. We assume that T is a power of 2, and denote $\log_2$ as simply $\log$. Suppose that we would like to support range queries on D different attributes, each of them can take on values in $[T_1], [T_2], \ldots, [T_D]$ respectively. We formally define a D-dimensional lattice, points and hyper-rectangles below.

Definition 3.1.1 (D-dimensional lattice, point, hyper-rectangle) *Let $\Delta = (T_1, T_2, \ldots, T_D)$. $\mathbb{L}_\Delta = [T_1] \times [T_2] \times \ldots \times [T_D]$ defines a D-**dimensional lattice**. A D-tuple $\mathbf{X} = (x_1, x_2, \ldots, x_D)$ defines a **point** in $\mathbb{L}_\Delta$, where $x_d \in [T_d] (\forall d \in [D])$. A **hyper-rectangle** $\mathbf{B}$ in $\mathbb{L}_\Delta$ is defined as $\mathbf{B}(s_1, t_1, s_2, t_2, \ldots, s_D, t_D) = \{(x_1, x_2, \ldots, x_D) \big| \forall d \in [D], x_d \in [s_d, t_d]\}$ ($\forall d \in [D], 1 \leq s_d \leq t_d \leq T_d$).*

An MRQED scheme consists of four (possibly randomized) polynomial-time algorithms: $\mathbf{Setup}$, $\mathbf{Encrypt}$, $\mathbf{DeriveKey}$ and $\mathbf{QueryDecrypt}$. In the network audit log example, an authority runs $\mathbf{Setup}$ to generate public parameters and a master private key; a gateway runs the $\mathbf{Encrypt}$ algorithm to encrypt a flow. Encryption is performed on a pair $(\mathbf{Msg}, \mathbf{X})$. The message $\mathbf{Msg}$ is an arbitrary string, and $\mathbf{X}$ is a point in multi-dimensional space, representing the attributes. For example, suppose that we would like to support queries on the following three attributes of a flow: time-stamp t, source address a, and destination port p. The tuple (t, a, p) then becomes the point $\mathbf{X}$, and the entire flow summary forms the message $\mathbf{Msg}$. Whenever necessary, the authority can run the $\mathbf{DeriveKey}$ algorithm, and compute a decryption key allowing the decryption of flows whose attributes fall within a certain range. Given this decryption key, an auditor runs the $\mathbf{QueryDecrypt}$ algorithm over the encrypted data to decrypt the relevant flows. We now formally define MRQED.

Definition 3.1.2 (MRQED) *An Multi-dimensional Range Query over Encrypted Data (MRQED) scheme consists of the following polynomial-time randomized algorithms.*

1. $\mathbf{Setup}(\Sigma, \mathbb{L}_\Delta)$: Takes a security parameter Σ and D-dimensional lattice $\mathbb{L}_\Delta$ and outputs public key PK and master private key SK.
2. $\mathbf{Encrypt}(\mathsf{PK}, \mathbf{X}, \mathbf{Msg})$: Takes a public key PK, a point $\mathbf{X}$, and a message $\mathbf{Msg}$ from the message space $\mathbb{M}$ and outputs a ciphertext $\mathbf{C}$.
3. $\mathbf{DeriveKey}(\mathsf{PK}, \mathsf{SK}, \mathbf{B})$: Takes a public key PK, a master private key SK, and a hyper-rectangle $\mathbf{B}$ and outputs decryption key for hyper-rectangle $\mathbf{B}$.

4. $\mathbf{QueryDecrypt}(\mathsf{PK}, \mathsf{DK}, \mathbf{C})$: Takes a public key PK, a decryption key DK, and a ciphertext $\mathbf{C}$ and outputs either a plaintext $\mathbf{Msg}$ or $\perp$, signaling decryption failure.

For each message $\mathbf{Msg} \in \mathbb{M}$, hyper-rectangle $\mathbf{B} \subseteq \mathbb{L}_\Delta$, and point $\mathbf{X} \in \mathbb{L}_\Delta$, the above algorithms must satisfy the following consistency constraints:

$$\mathbf{QueryDecrypt}(\mathsf{PK}, \mathsf{DK}, \mathbf{C}) = \begin{cases} \mathbf{Msg} & \text{if } \mathbf{X} \in \mathbf{B} \\ \perp & \text{w.h.p., if } \mathbf{X} \notin \mathbf{B} \end{cases} \tag{3.1}$$

where $\mathbf{C} = \mathbf{Encrypt}(\mathsf{PK}, \mathbf{X}, \mathbf{Msg})$ and $\mathsf{DK} = \mathbf{DeriveKey}(\mathsf{PK}, \mathsf{SK}, \mathbf{B})$.

3.1.3 Security Definitions

Suppose that during time $[t_1, t_2]$, there is an outbreak of a worm characteristic by the port number p_1. Now the trusted authority issues a key for the range $t \in [t_1, t_2]$ and $p = p_1$ to a research group who has been asked to study the worm behavior. With this key, the research group should be able to decrypt only flows whose time-stamp and port number fall within the given range. The privacy of all other flows should still be preserved. Informally, suppose that a computationally bounded adversary has obtained decryption keys for regions $\mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_q$. Now given a ciphertext $\mathbf{C} = \mathbf{Encrypt}(\mathsf{PK}, \mathbf{X}, \mathbf{Msg})$ such that $\mathbf{X} \notin \mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_q$, the adversary cannot learn $\mathbf{X}$ or $\mathbf{Msg}$ from $\mathbf{C}$. Of course, since the adversary fails to decrypt $\mathbf{C}$ using keys for regions $\mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_q$, the adversary inevitably learns that the point $\mathbf{X}$ encrypted does not fall within these regions. But apart from this fact, the adversary cannot learn more information about $\mathbf{X}$ or $\mathbf{Msg}$.

We now formalize this intuition into a *selective security* game for MRQED. In particular, we will prove the security of our construction under the *selective, match-revealing* model. Here, selective security notion is similar to the selective-ID security for IBE schemes [3, 14, 15]. As mentioned in Section 2.1.1, a stronger notion of security is match-concealing, adaptive security.

Below, we state the formal definition of security in the selective, match-revealing model. Note that the security definitions for MRQED can be inferred from the security definition for general predicate encryption given in Section 2.1.1. However, for clarity, we now state it again in the context of multi-dimensional range queries.

Definition 3.1.3 (MR-selective security) *An MRQED scheme is **selectively secure** in the **match-revealing** (MR) model if all polynomial-time adversaries have at most a negligible advantage in the selective security game defined below.*

- **Init**: The adversary submits two points $\mathbf{X}_0^*, \mathbf{X}_1^* \in \mathbb{L}_\Delta$ where it wishes to be challenged.
- **Setup**: The challenger runs the $\mathbf{Setup}(\mathbf{\Sigma}, \mathbb{L}_\mathbf{\Delta})$ algorithm to generate PK, SK. It gives PK to the adversary, keeping SK secret.
- **Phase 1**: The adversary adaptively issues decryption key queries for hyper-rectangles:

  $$\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_{q_0}$$

  Furthermore, $\mathbf{X}_0^*$ and $\mathbf{X}_1^*$ are not contained in any hyper-rectangles queried in this phase, i.e., for $0 < i \leq q_0$, $\mathbf{X}_0^* \notin \mathbf{B}_i$, and $\mathbf{X}_1^* \notin \mathbf{B}_i$.

- **Challenge**: The adversary submits two equal length messages $\mathbf{Msg}_0, \mathbf{Msg}_1 \in \mathbb{M}$. The challenger flips a random coin, b, and encrypts $\mathbf{Msg}_b$ under $\mathbf{X}^*_b$. The ciphertext is passed to the adversary.
- **Phase 2**: Phase 1 is repeated. The adversary adaptively issues decryption key queries for hyper-rectangles $\mathbf{B}_{q_0+1}, \mathbf{B}_{q_0+2}, \ldots, \mathbf{B}_q$. As before, all hyper-rectangles queried in this stage must not contain $\mathbf{X}^*_0$ and $\mathbf{X}^*_1$.
- **Guess**: The adversary outputs a guess b' of b.

An adversary $\mathcal{A}$'s advantage in the above game is defined as $\mathbf{Adv}_{\mathcal{A}}(\Sigma) = \left|\Pr[b = b'] - \frac{1}{2}\right|$.

3.2 A First Attempt to Construct MRQED

3.2.1 Trivial construction

We first give a trivial construction for one-dimensional range query over encrypted data. We refer to one-dimensional range query over encrypted data as MRQED^1 where the superscript represents the number of dimensions.

In the trivial MRQED^1 construction, we make use of any secure public key encryption scheme. We first generate $O(T^2)$ public-private key pairs, one for each range $[s, t] \subseteq [1, T]$. To encrypt a message $\mathbf{Msg}$ under a point x, we produce $O(T^2)$ ciphertexts, one for each range $[s, t] \subseteq [1, T]$. In particular, if $x \in [s, t]$, we encrypt $\mathbf{Msg}$ with public key $\mathsf{pk}_{s,t}$; otherwise, we encrypt an invalid message $\perp$ with $\mathsf{pk}_{s,t}$. The decryption key for any range $[s, t]$ is then $\mathsf{sk}_{s,t}$, the private key for $[s, t]$.

We now give a formal description of the above construction for one-dimensional range queries. Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ denote a secure public key encryption scheme. $\mathcal{K}, \mathcal{E}, \mathcal{D}$ represent the key generation, encryption and decryption algorithm respectively. We build a MRQED^1 scheme based on $\mathcal{AE}$ as below.

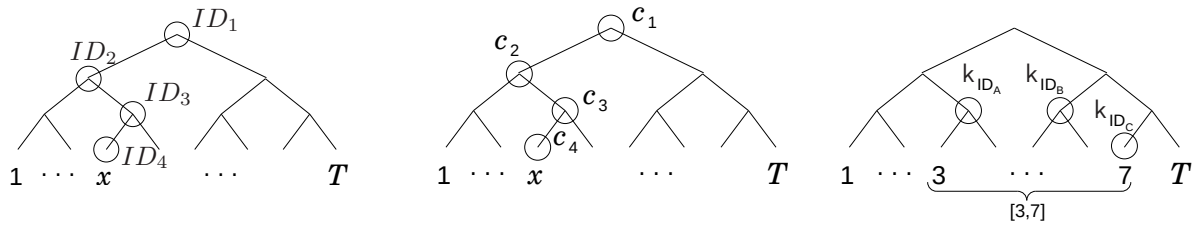
- During $\mathbf{Setup}$, one runs $\mathcal{K}$, the key generation algorithm, $O(T^2)$ times to generate the following public and private keys:

$$\mathsf{PK} = \left\{\mathsf{pk}_{s,t} \middle| 1 \le s \le t \le T\right\}, \quad \mathsf{SK} = \left\{\mathsf{sk}_{s,t} \middle| 1 \le s \le t \le T\right\}$$

- To encrypt a pair $(\mathbf{Msg}, x)$ where x is a point between 1 and T, first define for $1 \le s \le t \le T$

$$\delta_{s,t}(\mathbf{Msg}, x) = \begin{cases} \mathbf{Msg} & \text{if } s \le x \le t \\ \perp & \text{otherwise} \end{cases}$$

  where $\perp$ denotes the "invalid message". Now one runs the encryption algorithm $\mathcal{E}$, and for all ranges $[s, t] \subseteq [1, T]$, one encrypts $\delta_{s,t}(\mathbf{Msg}, x)$ under $\mathsf{pk}_{s,t}$. The result of encryption is a tuple of length T^2, denoted $(c_{1,1}, c_{1,2}, \ldots, c_{T,T})$.
- To release a decryption key $\mathbf{DK}_{s,t}$ for range $[s, t] \subseteq [1, T]$, one releases the key $\mathsf{sk}_{s,t}$.
- To decrypt a ciphertext $\mathbf{C} = (c_{1,1}, c_{1,2}, \ldots, c_{T,T})$ with $\mathbf{DK}_{s,t}$, one uses $\mathbf{DK}_{s,t}$ to decrypt $c_{s,t}$. Decryption either yields $\perp$, if the point x encrypted does not fall within the range $[s, t]$; or it yields the message $\mathbf{Msg}$, if x falls within $[s, t]$.



(a) The path from a leaf to the root.

(b) A ciphertext and a decryption key in MRQED^1.

Figure 3.1: An MRQED^1 scheme. ***(a)*** *Path from the leaf node representing $x \in [T]$ to the root. $\mathbb{P}(x) = \{ID_1, ID_2, ID_3, ID_4\}$.* ***(b)*** *Encryption under the point $x = 3$ and the keys released for the range $[3, 7]$.*

Clearly, the trivial MRQED^1 construction results in $O(T^2)$ public key size, $O(T^2)$ encryption overhead and ciphertext size, $O(1)$ decryption key size and $O(1)$ decryption cost.

One can easily extend the trivial construction into multiple dimensions. The resulting MRQED^D scheme requires that one encrypt $\delta_{\mathbf{B}}(\mathbf{Msg}, \mathbf{X})$ for all hyper-rectangles $\mathbf{B}$ in space. Therefore, the trivial MRQED^D scheme has $O(T^{2D})$ public key size, $O(T^{2D})$ encryption cost and ciphertext size, $O(1)$ decryption key size and $O(1)$ decryption cost.

3.2.2 Improved MRQED^1 construction based on AIBE

We show an improved MRQED construction based on Anonymous Identity-Based Encryption (AIBE). For clarity, we first explain the construction for one dimension. We call the scheme MRQED^1 where the superscript denotes the number of dimensions. We note that the primitives and notations introduced in this section will be used in our main construction.

Primitives: Efficient Representation of Ranges

To represent ranges efficiently, we build a binary interval tree over integers 1 through T.

Definition 3.2.1 (Interval tree) *Let $\mathsf{tr}(T)$ denote a binary interval tree over integers from 1 to T. Each node in the tree has a pre-assigned unique ID. For convenience, we define $\mathsf{tr}(T)$ to be the set of all node IDs in the tree. Each node in $\mathsf{tr}(T)$ represents a range. Let $\mathsf{cv}(ID)$ denote the range represented by node $ID \in \mathsf{tr}(T)$. Define $\mathsf{cv}(ID)$ as the following: Let ID be the i^{th} leaf node, then $\mathsf{cv}(ID) = i$. Otherwise, when ID is an internal node, let ID_1 and ID_2 denote its child nodes, then $\mathsf{cv}(ID) = \mathsf{cv}(ID_1) \cup \mathsf{cv}(ID_2)$. In other words, $\mathsf{cv}(ID)$ is the set of integers that correspond to the leaf descendants of ID.*

Given the interval tree $\mathsf{tr}(T)$, we define the $\mathbb{P}(x)$ of IDs covering a point $x \in [1, T]$, and the set $\Lambda(x)$ of IDs representing a range $[s, t] \subseteq [1, T]$.

- **Set of IDs covering a point x.** For a point $x \in [1, T]$ and some node $ID \in \mathsf{tr}(T)$, we say that ID *covers* the point x if $x \in \mathsf{cv}(ID)$. Define $\mathbb{P}(x)$ to be the set of IDs covering point x. Clearly, $\mathbb{P}(x)$ is the collection of nodes on the path from the root to the leaf node representing x. As an example, in Figure 3.1 (a), $\mathbb{P}(x) = \{ID_1, ID_2, ID_3, ID_4\}$.

- **Range as a collection of IDs.** A range $[s,t] \subseteq [1,T]$ is represented by a collection of nodes: $\Lambda(s,t) \subseteq \mathsf{tr}(T)$. We define $\Lambda(s,t)$ to be the smallest of all subsets $\mathbb{V} \subseteq \mathsf{tr}(T)$ such that $\bigcup_{ID \in \mathbb{V}} \mathsf{cv}(ID) = [s,t]$. It is not hard to see that for any $[s,t] \subseteq [1,T]$, $\Lambda(s,t)$ is uniquely defined, and its size $|\Lambda(s,t)|$ is at most $O(\log T)$.

We will make use of the following properties in our AIBE-based construction: If $x \in [s,t]$, then $\mathbb{P}(x) \cap \Lambda(s,t) \neq \emptyset$; in addition, $\mathbb{P}(x)$ and $\Lambda(s,t)$ intersect at only one node. Otherwise, if $x \notin [s,t]$, then $\mathbb{P}(x) \cap \Lambda(s,t) = \emptyset$.

AIBE-Based MRQED^1 Scheme

AIBE encrypts a message $\mathbf{Msg}$ using an identity ID as the public key. Given the private key for ID, one can successfully decrypt all messages encrypted by identity ID. The encryption scheme protects both the secrecy of the message $\mathbf{Msg}$ and the identity ID in the following sense: Given ciphertext $\mathbf{C}$, which is an encryption of $\mathbf{Msg}$ by identity ID_0, and given decryption keys for identities $ID_1, ID_2, \ldots, ID_q$ but not for ID_0, a computationally bounded adversary cannot learn anything about $\mathbf{Msg}$ or about ID_0 from the ciphertext $\mathbf{C}$. Researchers have successfully constructed secure AIBE schemes [1, 13] with $O(1)$ cost in all respects: in public parameter size, encryption cost, ciphertext size, decryption key size and decryption cost.

Given a secure AIBE scheme, we can construct an MRQED^1 scheme based on the following intuition. To encrypt the message $\mathbf{Msg}$ under point x, we encrypt $\mathbf{Msg}$ under all IDs in $\mathbb{P}(x)$. To release the decryption key for a range $[s,t] \subseteq [1,T]$, we release the keys for all IDs in $\Lambda(s,t)$. Now if $x \in [s,t]$, then $\mathbb{P}(x) \cap \Lambda(s,t) \neq \emptyset$. Suppose $\mathbb{P}(x)$ and $\Lambda(s,t)$ intersect at node ID. Then we can apply the decryption key at ID to the ciphertext encrypted under ID, and obtain the plaintext message $\mathbf{Msg}$. Otherwise, if $x \notin [s,t]$, then $\mathbb{P}(x) \cap \Lambda(s,t) = \emptyset$. In this case, the security of the underlying AIBE scheme ensures that a computationally bounded adversary cannot learn any information about the message $\mathbf{Msg}$ or the point x, except for the obvious fact (since decryption fails) that $x \notin [s,t]$.

Example. In Figure 3.1(b), we show a ciphertext $\mathbf{C}$ encrypted under the point x. Let $L = O(\log T)$ denote the height of the tree, $\mathbf{C}$ is composed of $O(\log T)$ components: $\{c_1, c_2, \ldots, c_L\}$. On the right, we show the decryption keys for the range $[3,7]$. Since $[3,7]$ can be represented by the set of nodes $\Lambda(3,7) = \{ID_A, ID_B, ID_C\}$, the decryption key for $[3,7]$ consists of three sub-keys, k_{ID_A}, k_{ID_B} and k_{ID_C}.

The AIBE-based construction has $O(1)$ public key size, $O(|\mathbb{P}(x)|)$ encryption cost and ciphertext size, and $O(|\Lambda(s,t)|)$ decryption key size. Since $|\mathbb{P}(x)| = O(\log T)$, and $|\Lambda(s,t)| = O(\log T)$, we get $O(\log T)$ in encryption cost, ciphertext size, and decryption key size. Later, we will show that decryption can be done in $O(\log T)$ time as well.

Stated more formally, given a secure AIBE scheme denoted:

$$\mathbf{Setup}^*(\Sigma),\ \mathbf{DeriveKey}^*(\mathsf{PK}, \mathsf{SK}, ID),\ \mathbf{Encrypt}^*(\mathsf{PK}, ID, \mathbf{Msg}),\ \mathbf{Decrypt}^*(\mathsf{PK}, \mathbf{DK}, \mathbf{C}),$$

one can construct a secure MRQED^1 scheme as below:

- $\mathbf{Setup}(\mathbf{\Sigma}, \mathbf{T})$ calls $\mathbf{Setup}^*(\Sigma)$ and outputs PK and SK.

- $\mathbf{Encrypt}(\mathsf{PK}, \mathbf{x}, \mathbf{Msg})$ encrypts the message $\mathbf{Msg}$ under every $ID \in \mathbb{P}(x)$. In other words, $\mathbf{Encrypt}$ yields $\mathbf{C} = \{c_{ID} | ID \in \mathbb{P}(x)\}$, where $c_{ID} = \mathbf{Encrypt}^*(\mathsf{PK}, ID, \mathbf{Msg}||0^{m'})$. To check whether a decryption is valid, prior to encryption, we append m' trailing 0s denoted $0^{m'}$ to message $\mathbf{Msg} \in \{0,1\}^m$.
- $\mathbf{DeriveKey}(\mathsf{PK}, \mathsf{SK}, [s,t])$ releases a decryption key k_{ID} for each $ID \in \Lambda(s,t)$. k_{ID} is computed as $\mathsf{k}_{ID} = \mathbf{DeriveKey}^*(\mathsf{PK}, \mathsf{SK}, ID)$. The entire decryption key for the range $[s,t]$ is then the set $\mathbf{DK}_{s,t} = \{\mathsf{k}_{ID} \mid ID \in \Lambda(s,t)\}$.
- $\mathbf{QueryDecrypt}(\mathsf{PK}, \mathbf{DK}, \mathbf{C})$ tries each key $\mathsf{k}_{ID} \in \mathbf{DK}_{s,t}$ on each ciphertext $c_{ID'} \in \mathbf{C}$. If $ID = ID'$, then $\mathbf{Decrypt}^*(\mathsf{PK}, \mathsf{k}_{ID}, c_{ID'})$ yields result of the form $\widehat{\mathbf{Msg}}||0^{m'}$. In this case, we accept the result and exit the $\mathbf{QueryDecrypt}$ algorithm. If all trials fail to yield result of the form $\widehat{\mathbf{Msg}}||0^{m'}$, $\mathbf{QueryDecrypt}$ outputs $\perp$, indicating failure to decrypt.

Note that in the AIBE-based construction, if we simply try all decryption keys over all ciphertexts, then decryption would require $O(|\mathbb{P}(x)| \cdot |\Lambda(s,t)|)$ time; since $|\mathbb{P}(x)| = O(\log T), |\Lambda(s,t)| = O(\log T)$, decryption would require $O(\log^2 T)$ time. However, observe that it is not necessary to try k_{ID} on $c_{ID'}$, if ID and ID' are at different depth in the tree; since then, ID and ID' cannot be equal. Thus we only need to try k_{ID} on $c_{ID'}$ if ID and ID' are at the same depth in the tree, which requires knowledge of the depth of ID' for ciphertext $c_{ID'}$. Of course, we cannot directly release ID' for ciphertext $c_{ID'}$, since the encryption is meant to hide ID'. However, since each ciphertext $\mathbf{C}$ has a portion at every depth of the tree, we can give out the depth of ID' for each $c_{ID'} \in \mathbf{C}$ without leaking any information about ID'. In this way, we reduce the decryption cost to $O(\log T)$ rather than $O(\log^2 T)$.

We emphasize that using AIBE as the underlying encryption scheme is crucial to ensuring the security of the derived MRQED[1] scheme. In particular, a non-anonymous IBE scheme is not suitable to use as the underlying encryption scheme, since IBE hides only the message $\mathbf{Msg}$ but not the attribute x.

3.2.3 AIBE-Based MRQED^D Construction

The same idea can be applied to construct an MRQED^D scheme, resulting in $O(1)$ public key size, $O\left((\log T)^D\right)$ encryption cost, ciphertext size, decryption key size, and decryption cost. The details of this construction is not crucial to the understanding of our main construction. However, in describing this construction, we highlight a few important definitions, including the notion of a simple hyper-rectangle, and the definition of $\Lambda^{\times}(\mathbf{B})$. These definitions will later be used in our main construction.

We build D binary interval trees, one for each dimension. We assign a globally unique ID to each node in the D trees.

Representing a hyper-rectangle. We represent an arbitrary hyper-rectangle as a collection of *simple hyper-rectangles*. To illustrate this idea, we first give a formal definition of a simple hyper-rectangle, and then state how to represent an arbitrary hyper-rectangle as a collection of simple hyper-rectangles. Simply put, a simple hyper-rectangle is a hyper-rectangle $\mathbf{B}_0$ in space, such that

$\mathbf{B}_0$ can be represented by a single node in the tree of every dimension. More specifically, a hyper-rectangle $\mathbf{B}(s_1, t_1, \ldots, s_D, t_D)$ in space is composed of a range along each dimension. If for all $1 \leq d \leq D$, $|\Lambda(s_d, t_d)| = 1$, i.e., $[s_d, t_d]$ is a simple range in the d^{th} dimension, then we say that the hyper-rectangle $\mathbf{B}(s_1, t_1, \ldots, s_D, t_D)$ is a *simple hyper-rectangle*. A simple hyper-rectangle can be defined by a single node from each dimension. We can assign a unique identity to each simple-rectangle $\mathbf{B}_0(s_1, t_1, \ldots, s_D, t_D)$ in space. Define

$$\mathsf{id}_{\mathbf{B}_0} = (ID_1, ID_2, \ldots, ID_D),$$

where $ID_d (1 \leq i \leq D)$ is the node representing $[s_d, t_d]$ in the d^{th} dimension.

Definition 3.2.2 (Hyper-rectangle as a collection of simple hyper-rectangles) *Given an hyper-rectangle $\mathbf{B}(s_1, t_1, \ldots, s_D, t_D)$, denote $\Lambda_d(\mathbf{B}) := \Lambda(s_d, t_d)$ for $d \in [D]$. $\Lambda(\mathbf{B})$ is the collection of nodes representing range $[s_d, t_d]$ in the d^{th} dimension. The hyper-rectangle $\mathbf{B}$ can be represented as a collection $\Lambda^{\times}(\mathbf{B})$ of simple hyper-rectangles:*

$$\Lambda^{\times}(\mathbf{B}) = \Lambda_1(\mathbf{B}) \times \Lambda_2(\mathbf{B}) \times \ldots \times \Lambda_D(\mathbf{B})$$

In particular, for every $\mathsf{id} \in \Lambda^{\times}(\mathbf{B})$, id is a vector of the form $(ID_1, ID_2, \ldots, ID_D)$, where ID_d $(d \in [D])$ is a node in the tree corresponding to the d^{th} dimension. Therefore, id uniquely specifies a simple hyper-rectangle $\mathbf{B}_0$ in space.

Clearly, $|\Lambda^{\times}(\mathbf{B})| = O\left((\log T)^D\right)$; in addition, $\Lambda^{\times}(\mathbf{B})$ can be efficiently computed. Given the above definitions, we briefly describe the AIBE-based MRQED^D construction.

Encryption. Suppose that now we would like to encrypt a message $\mathbf{Msg}$ and the point $\mathbf{X} = (x_1, x_2, \ldots, x_D)$. We encrypt the message $\mathbf{Msg}$ under all simple hyper-rectangles that contain the point $\mathbf{X} = (x_1, x_2, \ldots, x_D)$. This is equivalent to encrypting $\mathbf{Msg}$ under the cross-product of D different paths to the root. Specifically, for $d \in [D]$, denote $\mathbb{P}_d(\mathbf{X}) := \mathbb{P}(x_d)$. $\mathbb{P}_d(\mathbf{X})$ is the path from the root to the leaf node representing x_d in the d^{th} dimension. Define the cross-product of all D different paths to the root:

$$\mathbb{P}^{\times}(\mathbf{X}) = \mathbb{P}_1(\mathbf{X}) \times \mathbb{P}_2(\mathbf{X}) \times \ldots \times \mathbb{P}_D(\mathbf{X}).$$

Then, to encrypt $\mathbf{Msg}$ and $\mathbf{X}$, we use AIBE to encrypt $\mathbf{Msg}$ under every $\mathsf{id} \in \mathbb{P}^{\times}(\mathbf{X})$. Since $|\mathbb{P}^{\times}(\mathbf{X})| = O\left((\log T)^D\right)$, both encryption cost and ciphertext size are $O\left((\log T)^D\right)$.

Key derivation and decryption. To issue decryption keys for a hyper-rectangle $\mathbf{B}$, we issue a key for every $\mathsf{id} \in \Lambda^{\times}(\mathbf{B})$. Since $|\Lambda^{\times}(\mathbf{B})| = O\left((\log T)^D\right)$, the decryption key has size $O\left((\log T)^D\right)$. Now if $\mathbf{X} \in \mathbf{B}$, then $\mathbb{P}^{\times}(\mathbf{X}) \cap \Lambda^{\times}(\mathbf{B}) \neq \emptyset$; in addition, $\mathbb{P}^{\times}(\mathbf{X})$ and $\Lambda^{\times}(\mathbf{B})$ intersect at exactly one simple hyper-rectangle $\mathsf{id}_{\mathbf{B}_0}$, where the keys and the ciphertexts overlap. In this case, we use the key for $\mathsf{id}_{\mathbf{B}_0}$ to decrypt the ciphertext for $\mathsf{id}_{\mathbf{B}_0}$. Otherwise, if $\mathbf{X} \notin \mathbf{B}$, then $\mathbb{P}^{\times}(\mathbf{X}) \cap \Lambda^{\times}(\mathbf{B}) = \emptyset$. In this case, the security of the underlying AIBE schemes ensures the security of the MRQED^D constructions.

3.3 The Main MRQED Construction

In Section 3.2.3, we showed an AIBE-based MRQED^D construction with $O(1)$ public key size, $O\left((\log T)^D\right)$ encryption cost and ciphertext size, $O\left((\log T)^D\right)$ decryption key size and decryption cost. In this section, we propose a new MRQED^D construction with $O\left(D \log T\right)$ public key size, $O\left(D \log T\right)$ encryption cost and ciphertext size, $O\left(D \log T\right)$ decryption key size, and $O\left((\log T)^D\right)$ decryption cost.

Our main MRQED construction is relies on bilinear groups of prime order. Therefore, we begin by giving some background knowledge on pairing and bilinear groups.

3.3.1 Background on bilinear groups

A pairing is an efficiently computable, non-degenerate function, $\mathbf{e} : \mathbb{G} \times \widehat{\mathbb{G}} \rightarrow \mathbb{G}'$, satisfying the bilinear property that $\mathbf{e}(g^r, \widehat{g}^s) = \mathbf{e}(g, \widehat{g})^{rs}$. $\mathbb{G}$, $\widehat{\mathbb{G}}$ and $\mathbb{G}'$ are all groups of prime order. g, $\widehat{g}$ and $\mathbf{e}(g, \widehat{g})$ are generators of $\mathbb{G}$, $\widehat{\mathbb{G}}$ and $\mathbb{G}'$ respectively. Although our MRQED scheme can be constructed using asymmetric pairing, for simplicity, we describe our scheme using symmetric pairing in the remainder of this thesis proposal, i.e., $\mathbb{G} = \widehat{\mathbb{G}}$.

We name a tuple $\mathbf{G} = [p, \mathbb{G}, \mathbb{G}', g, \mathbf{e}]$ a bilinear instance, where $\mathbb{G}$ and $\mathbb{G}'$ are two cyclic groups of prime order p. We assume an efficient generation algorithm that on input of a security parameter Σ, outputs $\mathbf{G} \stackrel{R}{\leftarrow} \text{Gen}(\Sigma)$ where $\log_2 p = \Theta(\Sigma)$.

We rely on the following complexity assumptions:

Decision BDH Assumption : The Decision Bilinear DH assumption, first used by Joux [27], later used by IBE systems [8], posits the hardness of the following problem: Given

$$[g, g^{z_1}, g^{z_2}, g^{z_3}, Z] \in \mathbb{G}^4 \times \mathbb{G}'$$

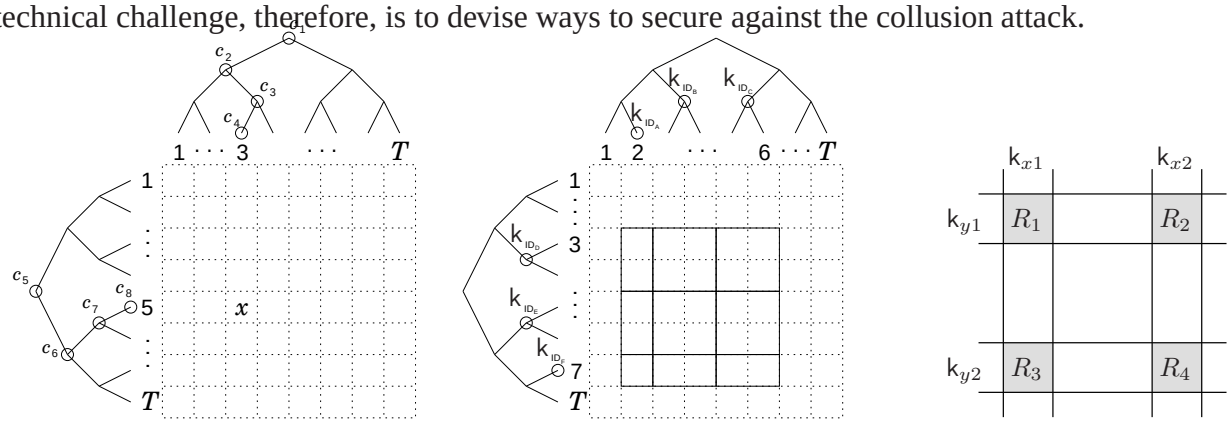
where exponents z_1, z_2, z_3 are picked at random from $\mathbb{Z}_p$, decide whether $Z = \mathbf{e}(g, g)^{z_1 z_2 z_3}$.

Decision Linear Assumption : The Decision Linear assumption, first proposed by Boneh, Boyen and Shacham for group signatures [5], posits the hardness of the following problem: Given $[g, g^{z_1}, g^{z_2}, g^{z_1 z_3}, g^{z_2 z_4}, Z] \in \mathbb{G}^6$, where z_1, z_2, z_3, z_4 are picked at random from $\mathbb{Z}_p$, decide whether $Z = g^{z_3 + z_4}$.

3.3.2 Intuition

We build D interval trees over integers from 1 to T, each representing a separate dimension. Assume each tree node has a globally unique ID. In the previous section, we showed a naive construction for MRQED^D based on AIBE. This naive construction encrypts $\mathbf{Msg}$ under the $O((\log T)^D)$ simple hyper-rectangles that contain the point $\mathbf{X}$; and releases decryption keys for the $O((\log T)^D)$ simple hyper-rectangles that compose a hyper-rectangle $\mathbf{B}$. Our goal is to reduce the ciphertext size and decryption key size to $O(D \log T)$ instead. However, as we will soon

explain, naively doing this introduces the *collusion attack* as shown in Figure 3.2 (b). Our main technical challenge, therefore, is to devise ways to secure against the collusion attack.



(a) A ciphertext and a decryption key in MRQED^2. (b) Collusion.

Figure 3.2: An MRQED^2 scheme. ***(a)*** *Encryption under the point $x = (3, 5)$ and the keys released for the range $[2, 6] \times [3, 7]$.* ***(b)*** *With decryption keys $\mathsf{k}_{x1}, \mathsf{k}_{y1}$ for region R_1 and $\mathsf{k}_{x2}, \mathsf{k}_{y2}$ for region R_4, regions R_2 and R_3 are compromised.*

Reducing the ciphertext size. In other words, rather than encryption $\mathbf{Msg}$ for each simple hyper-rectangle in $\mathbb{P}^{\times}(\mathbf{X}) = \mathbb{P}_1(\mathbf{X}) \times \ldots \times \mathbb{P}_D(\mathbf{X})$, we would like to encrypt $\mathbf{Msg}$ for each tree node in the the union of these D different paths:

$$\mathbb{P}^{\cup}(\mathbf{X}) = \mathbb{P}_1(\mathbf{X}) \cup \ldots \cup \mathbb{P}_D(\mathbf{X}).$$

Reducing the decryption key size. Instead of representing an arbitrary hyper-rectangle using the collection of simple hyper-rectangles, we can represent a simple hyper-rectangle $\mathbf{B}$ as the collection of disjoint intervals over different dimensions:

Definition 3.3.1 (Hyper-rectangle as a collection of nodes) *A hyper-rectangle $\mathbf{B} \subseteq \mathbb{L}_{\Delta}$ gives a collection of nodes corresponding to disjoint intervals over different dimensions:*

$$\Lambda^{\cup}(\mathbf{B}) = \Lambda_1(\mathbf{B}) \cup \Lambda_2(\mathbf{B}) \cup \ldots \cup \Lambda_D(\mathbf{B})$$

Note that for all hyper-rectangle $\mathbf{B} \subseteq \mathbb{L}_{\Delta}$, $|\Lambda^{\cup}(\mathbf{B})| = O(D \log T)$; in addition, $\Lambda^{\cup}(\mathbf{B})$ can be computed efficiently.

With the above definition, rather than releasing keys for each simple hyper-rectangle in $\Lambda^{\times}(\mathbf{B}) = \Lambda_1(\mathbf{B}) \times \ldots \times \Lambda_D(\mathbf{B})$, we would like to release keys for each ID in $\Lambda_1(\mathbf{B}) \cup \ldots \cup \Lambda_D(\mathbf{B})$.

Example. Figure 3.2 (a) is an example in two dimensions. To encrypt under the point $(3, 5)$, we find the path from the leaf node 3 to the root in the first dimension, and the path from the leaf node 5 to the root in the second dimension. We then produce a block in the ciphertext corresponding to each node on the two paths. In the first dimension, we produce blocks c_1, c_2, c_3 and c_4. In the second dimension, we produce blocks c_5, c_6, c_7 and c_8. To release decryption keys for the range

$[2,6] \times [3,7]$, we find a collection $\Lambda(2,6)$ of nodes covering the range $[2,6]$ in the first dimension; and a collection $\Lambda(3,7)$ of nodes covering $[3,7]$ in the second dimension. We issue a block in the decryption key corresponding to each node in $\Lambda(2,6)$ and in $\Lambda(3,7)$. In the first dimension, we create blocks $\mathsf{k}_{ID_A}, \mathsf{k}_{ID_B}$, and k_{ID_C}; and in the second dimension, we create blocks $\mathsf{k}_{ID_D}, \mathsf{k}_{ID_E}$, and k_{ID_F}.

Preventing the collusion attack. Unfortunately, naively doing the above is equivalent to applying the AIBE-based MRQED[1] scheme independently in each dimension. As we demonstrate in Figure 3.2 (b), such a scheme is susceptible to the collusion attack. Suppose that Figure 3.2 (b), every rectangle is a simple rectangle. Now suppose that an adversary were given the decryption keys for region R_1 and R_4, then the adversary would have collected keys $\mathsf{k}_{R1} = \{\mathsf{k}_{x1}, \mathsf{k}_{y1}\}$, $\mathsf{k}_{R4} = \{\mathsf{k}_{x2}, \mathsf{k}_{y2}\}$. With these, the adversary would be able to reconstruct the keys for R_2 and R_3: $\mathsf{k}_{R2} = \{\mathsf{k}_{x2}, \mathsf{k}_{y1}\}$, $\mathsf{k}_{R3} = \{\mathsf{k}_{x1}, \mathsf{k}_{y2}\}$. Hence, our major challenge is to find a way to secure against the collusion attack without incurring additional cost. We use a *binding* technique to prevent the collusion attack: we use re-randomization to tie together the sub-keys in different dimensions. For example, in Figure 3.2 (b), when we release the decryption key for region R_1, instead of releasing $\{\mathsf{k}_{x1}, \mathsf{k}_{y1}\}$, we release $\{\widetilde{\mu}_x \mathsf{k}_{x1}, \widetilde{\mu}_y \mathsf{k}_{y1}\}$, where $\widetilde{\mu}_x$ and $\widetilde{\mu}_y$ are random numbers that we pick each time we issue a decryption key. Likewise, when releasing the key for region R_4, we release $\{\widetilde{\mu}'_x \mathsf{k}_{x2}, \widetilde{\mu}'_y \mathsf{k}_{y2}\}$, where $\widetilde{\mu}'_x$ and $\widetilde{\mu}'_y$ are two random numbers picked independently from $\widetilde{\mu}_x$ and $\widetilde{\mu}_y$. Of course, in the real construction, $\widetilde{\mu}_x$ and $\widetilde{\mu}_y$ ($\widetilde{\mu}'_x$ and $\widetilde{\mu}'_y$) also need to satisfy certain algebraic properties (e.g., $\widetilde{\mu}_x \widetilde{\mu}_y = \widetilde{\mu}'_x \widetilde{\mu}'_y =$ some invariant) to preserve the internal consistency of our scheme. In this way, components in the decryption key for R_1 cannot be used in combination with components in the decryption key for R_4.

3.3.3 The Main Construction

We are now ready to describe our construction. Define $L = O(\log T)$ to represent the height of a tree. Assume that node IDs are picked from $\mathbb{Z}_p^*$. We append a message $\mathbf{Msg} \in \{0,1\}^m$ with a series of trailing zeros, $0^{m'}$, prior to encryption. Assume that $\{0,1\}^{m+m'} \subseteq \mathbb{G}'$.

$\mathbf{Setup}(\mathbf{\Sigma}, \mathbb{L}_{\mathbf{\Delta}})$ To generate public parameters and the master private key, the setup algorithm first generates a bilinear instance $\mathbf{G} = [p, \mathbb{G}, \mathbb{G}', g, \mathbf{e}] \xleftarrow{R} \mathrm{Gen}(\Sigma)$. Then, the setup algorithm does the following.

1. Select at random the following parameters from $\mathbb{Z}_p^{8DL+1}$:

$$\omega, \left[\alpha_{\varphi,1}, \alpha_{\varphi,2}, \beta_{\varphi,1}, \beta_{\varphi,2}, \theta_{\varphi,1}, \theta_{\varphi,2}, \theta'_{\varphi,1}, \theta'_{\varphi,2}\right]_{\substack{\varphi=(d,l) \\ \in [D]\times[L]}}$$

   In addition, we require that the α's and the β's be forcibly non-zero. At this point, we give a brief explanation of our notation. The variable φ is used to index a tuple $(d,l) \in [D] \times [L]$, where d denotes the dimension and l denote the depth of a node in the corresponding tree.

2. Publish $\mathbf{G}$ and the following public parameters $\mathsf{PK} \in \mathbb{G}' \times \mathbb{G}^{8DL}$:

$$\Omega \leftarrow \mathbf{e}(g,g)^{\omega},$$
$$\left[\begin{array}{l} a_{\varphi,1} \leftarrow g^{\alpha_{\varphi,1}\theta_{\varphi,1}},\ a_{\varphi,2} \leftarrow g^{\alpha_{\varphi,2}\theta_{\varphi,2}}, \\ a'_{\varphi,1} \leftarrow g^{\alpha_{\varphi,1}\theta'_{\varphi,1}},\ a'_{\varphi,2} \leftarrow g^{\alpha_{\varphi,2}\theta'_{\varphi,2}}, \\ b_{\varphi,1} \leftarrow g^{\beta_{\varphi,1}\theta_{\varphi,1}},\ b_{\varphi,2} \leftarrow g^{\beta_{\varphi,2}\theta_{\varphi,2}}, \\ b'_{\varphi,1} \leftarrow g^{\beta_{\varphi,1}\theta'_{\varphi,1}},\ b'_{\varphi,2} \leftarrow g^{\beta_{\varphi,2}\theta'_{\varphi,2}}, \end{array}\right]_{\substack{\varphi=(d,l)\in \\ [D]\times[L]}}$$

3. Retain a master private key $\mathsf{SK} \in \mathbb{G}^{8DL+1}$ comprising the following elements:

$$\widetilde{\omega} \leftarrow g^{\omega},$$
$$\left[\begin{array}{ll} \mathsf{a}_{\varphi,1} \leftarrow g^{\alpha_{\varphi,1}}, & \mathsf{a}_{\varphi,2} \leftarrow g^{\alpha_{\varphi,2}}, \\ \mathsf{b}_{\varphi,1} \leftarrow g^{\beta_{\varphi,1}}, & \mathsf{b}_{\varphi,2} \leftarrow g^{\beta_{\varphi,2}}, \\ y_{\varphi,1} \leftarrow g^{\alpha_{\varphi,1}\beta_{\varphi,1}\theta_{\varphi,1}}, & y_{\varphi,2} \leftarrow g^{\alpha_{\varphi,2}\beta_{\varphi,2}\theta_{\varphi,2}}, \\ y'_{\varphi,1} \leftarrow g^{\alpha_{\varphi,1}\beta_{\varphi,1}\theta'_{\varphi,1}}, & y'_{\varphi,2} \leftarrow g^{\alpha_{\varphi,2}\beta_{\varphi,2}\theta'_{\varphi,2}} \end{array}\right]_{\substack{\varphi=(d,l) \\ \in[D]\times[L]}}$$

Notice that in the public parameters and the master key, we have different versions of the same variable, e.g., $a_{\varphi,1}, a_{\varphi,2}, a'_{\varphi,1}, a'_{\varphi,2}$. Although they seem to be redundant, they are actually needed to provide sufficient degrees of randomness for our proof to go through. The reasons for having these different versions will become clear once the reader has gone over the detailed proof provided in Section 3.7.

$\mathbf{DeriveKey}(\mathsf{PK}, \mathsf{SK}, \mathbf{B})$ The following steps compute the decryption key for hyper-rectangle $\mathbf{B}$, given public key PK and master private key SK.

1. Pick $O(D \cdot L)$ random integers from $\mathbb{G}^D \times \mathbb{Z}_p^{2|\Lambda^{\cup}(\mathbf{B})|}$:

$$\left[\widetilde{\mu}_d\right]_{d\in[D]}, \quad \left[\lambda_{ID,1}, \lambda_{ID,2}\right]_{ID\in\Lambda^{\cup}(\mathbf{B})}$$

such that $\prod_{d\in[D]} \widetilde{\mu}_d = \widetilde{\omega}$. The reason for having an overhead tilde for the variable $\widetilde{\mu}_d$ is to associate it with the variable $\widetilde{\omega}$, since they both belong to the group $\mathbb{G}$, and they satisfy the condition that $\prod_{d\in[D]} \widetilde{\mu}_d = \widetilde{\omega}$. We note that the random $\widetilde{\mu}_d$'s generated in this stage are later used to re-randomize the components of the decryption key. In this way, components in different dimensions are tied to each other; and components from one decryption key cannot be used in combination with components from another decryption key. This is how we prevent the collusion attack as shown in Figure 3.2 (b).

2. Compute and release a decryption key $\mathbf{DK} \in \mathbb{G}^{5|\Lambda^{\cup}(\mathbf{B})|}$. $\mathbf{DK}$ is composed of a portion $\mathbf{DK}(\mathbf{ID})$ for each $ID \in \Lambda^{\cup}(\mathbf{B})$. In the following definition for $\mathbf{DK}(ID)$, $\varphi = (d,l) = \Phi(ID)$ represents the dimension and depth of node ID; without risk of ambiguity, denote $\lambda_1 = \lambda_{ID,1}, \lambda_2 = \lambda_{ID,2}$. $\mathbf{DK}(ID)$ is defined below:

$$\widetilde{\mu}_d \left(y_{\varphi,1}^{ID} y'_{\varphi,1}\right)^{\lambda_1} \left(y_{\varphi,2}^{ID} y'_{\varphi,2}\right)^{\lambda_2},\ \mathsf{a}_{\varphi,1}^{-\lambda_1},\ \mathsf{b}_{\varphi,1}^{-\lambda_1},\ \mathsf{a}_{\varphi,2}^{-\lambda_2},\ \mathsf{b}_{\varphi,2}^{-\lambda_2}$$

Observe that we release a portion of the decryption key for each node in $\Lambda^{\cup}(\mathcal{B})$, as opposed to for each hyper-rectangle in $\Lambda^{\times}(\mathcal{B})$. In this way, the size of the private key is $O(DL)$, instead of $O(L^D)$. Also observe that we multiply the first element of $\mathbf{DK}(ID)$ by $\widetilde{\mu}_d$. This illustrates the *binding* technique used to tie together components in different dimensions. In this way, components in one decryption key cannot be used in combination with components in another decryption key; therefore, we successfully prevent the collusion attack.

$\mathbf{Encrypt}(\mathsf{PK}, \mathbf{X}, \mathbf{Msg})$ We create a block in the ciphertext for every $ID \in \mathbb{P}^{\cup}(\mathbf{X})$. Equivalently, for each dimension d and depth l, denote $\varphi = (d, l)$, we create a portion of the ciphertext corresponding to the node $\mathcal{I}_\varphi$, residing in the d^{th} tree at depth l, on the path $\mathbb{P}_d(\mathbf{X})$ to the root. We now describe the $\mathbf{Encrypt}$ algorithm in the following steps:

1. Select $2DL + 1$ random integers: select $r \in_R \mathbb{Z}_p$, select $[r_{\varphi,1}, r_{\varphi,2}]_{\varphi=(d,l)\in[D]\times[L]} \in_R \mathbb{Z}_p^{2DL}$.
2. For $\varphi = (d, l) \in [D] \times [L]$, define $\mathcal{I}_\varphi = \mathcal{I}_\varphi(\mathbf{X})$, i.e., the node at depth l in $\mathbb{P}_d(\mathbf{X})$ in the d^{th} dimension. Now compute and output the following ciphertext $\mathbf{C} \in \mathbb{G}' \times \mathbb{G}^{4DL+1}$:

$$\left[\begin{array}{l} (\mathbf{Msg}||0^{m'}) \cdot \Omega^{-r},\ \ g^r, \\ (b_{\varphi,1}{}^{\mathcal{I}_\varphi} b'_{\varphi,1})^{r_{\varphi,1}},\ (a_{\varphi,1}{}^{\mathcal{I}_\varphi} a'_{\varphi,1})^{r-r_{\varphi,1}}, \\ (b_{\varphi,2}{}^{\mathcal{I}_\varphi} b'_{\varphi,2})^{r_{\varphi,2}},\ (a_{\varphi,2}{}^{\mathcal{I}_\varphi} a'_{\varphi,2})^{r-r_{\varphi,2}} \end{array}\right]_{\substack{\varphi=(d,l)\in \\ [D]\times[L]}}$$

$\mathbf{QueryDecrypt}(\mathsf{PK}, \mathbf{DK}, \mathbf{C})$ We first give an overview on how $\mathbf{QueryDecrypt}$ works. Recall that a decryption key $\mathbf{DK} = \{\mathbf{DK}(ID) \mid ID \in \Lambda^{\cup}(\mathbf{B})\}$ is composed of a portion $\mathbf{DK}(ID)$ for each $ID \in \Lambda^{\cup}(\mathbf{B})$. We now reconstruct a decryption key for each simple hyper-rectangle $\mathsf{id}_{\mathbf{B}_0} \in \Lambda^{\times}(\mathbf{B})$ as below. We grab from $\mathbf{DK}$ a sub-key from each dimension: for each $d \in [D]$, grab a sub-key $\mathbf{DK}(ID_d)$ from the d^{th} dimension, where $ID_d \in \Lambda_d(\mathbf{B})$. The collection of sub-keys $\{\mathbf{DK}(ID_1), \mathbf{DK}(ID_2), \ldots, \mathbf{DK}(ID_D)\}$ can now be jointly used to decrypt a message encrypted under the simple hyper-rectangle $\mathsf{id}_{\mathbf{B}_0} = (ID_1, \ldots, ID_D)$.

We also need to find the correct blocks in the ciphertext to apply this key for $\mathsf{id}_{\mathbf{B}_0}$. Recall that the ciphertext is of the form $\mathbf{C} = \left(c, c_0, [c_{\varphi,1}, c_{\varphi,2}, c_{\varphi,3}, c_{\varphi,4}]_{\varphi=(d,l)\in[D]\times[L]}\right)$. For convenience, denote $c_\varphi := [c_{\varphi,1}, c_{\varphi,2}, c_{\varphi,3}, c_{\varphi,4}]$ for $\varphi = (d, l) \in [D] \times [L]$. c_φ is the block in the ciphertext corresponding to a node in the d^{th} dimension and at depth l of the tree. Define $\Phi(ID) := (d, l)$ to extract the dimension and depth of the node ID. Now for a sub-key $\mathbf{DK}(ID)$, define $\varphi = \Phi(ID)$, it is not hard to see that $\mathbf{DK}(ID)$ should be used in combination with the block c_φ in the ciphertext.

The following algorithm iterates through the simple hyper-rectangles in $\Lambda^{\times}(\mathbf{B})$ and checks if the ciphertext can decrypt to a valid message under each simple hyper-rectangle in $\Lambda^{\times}(\mathbf{B})$.

For each simple hyper-rectangle $\Lambda^{\times}(\mathbf{B}_0) = \{(ID_1, ID_2, \ldots, ID_D)\} \subseteq \Lambda^{\times}(\mathbf{B})$,

(1) Let $\mathbf{DK}(ID_d) = (\mathsf{k}_{ID_d,0}, \mathsf{k}_{ID_d,1}, \mathsf{k}_{ID_d,2}, \mathsf{k}_{ID_d,3}, \mathsf{k}_{ID_d,4})$ represent the element in $\mathbf{DK}$ for ID_d, where $d \in [D]$.

(2) Try to decrypt $\mathbf{C}$ under $\mathbf{B}_0$ with the collection $\{\mathbf{DK}(ID_1), \mathbf{DK}(ID_2), \ldots, \mathbf{DK}(ID_D)\}$ of

sub-keys:

$$V \leftarrow c \cdot \prod_{\substack{d \in [D], \\ \varphi_d = \Phi(ID_d)}} \left[\mathbf{e}(c_0, \mathsf{k}_{ID_d,0}) \cdot \mathbf{e}(c_{\varphi_d,1}, \mathsf{k}_{ID_d,1}) \cdot \mathbf{e}(c_{\varphi_d,2}, \mathsf{k}_{ID_d,2}) \cdot \mathbf{e}(c_{\varphi_d,3}, \mathsf{k}_{ID_d,3}) \cdot \mathbf{e}(c_{\varphi_d,4}, \mathsf{k}_{ID_d,4}) \right]$$

If V is of the form $\widehat{\mathbf{Msg}}||0^{m'}$, then output $\widehat{\mathbf{Msg}}$ as the decrypted plaintext and exit.
If for all simple hyper-rectangles in $\Lambda^{\times}(\mathbf{B})$, the previous step fails to produce the plaintext, then output $\perp$.

When done naively, the above $\mathbf{QueryDecrypt}$ algorithm takes $O(D(\log T)^D)$ time. However, if one saves intermediate results, it can be done in $O((\log T)^D)$ time with $O(D \log T)$ storage. The above numbers takes into account all group operations, include multiplication, exponentiation and bilinear pairing. However, since a pairing operation is typically more expensive than exponentiation (and far more expensive than multiplication) in known bilinear groups, we are particularly interested in reducing the number of pairings at time of decryption. Notice that we can precompute all pairings $\mathbf{e}(c_0, \mathsf{k}_{ID_d,0})$ and pairings $\mathbf{e}(c_{\varphi_d,i}, \mathsf{k}_{ID_d,i})$ for $1 \leq i \leq 4$, and store the results in a look-up table. Therefore, the decryption algorithm requires $O(D \log T)$ pairings in total.

3.3.4 Consistency, Security

The following two theorems state the consistency and security of our MRQED construction.

Theorem 3.3.2 (Internal consistency) *The above defined MRQED construction satisfies the consistency requirement posed by Equation (3.1).*

Theorem 3.3.3 (Selective security) *The above defined MRQED construction is selectively secure against polynomial-time adversaries.*

Below we give an overview of the techniques used in the security proof. The detailed proofs of Theorem 3.3.2 and Theorem 3.3.3 are provided in Section 3.7 To prove the selective security of our MRQED^D construction, we decompose the selective MRQED game into two games: a selective confidentiality game and a selective anonymity game. By the hybrid argument, if no polynomial-time adversary has more than negligible advantage in either the confidentiality game or the anonymity game, then no polynomial-time adversary has more than negligible advantage in the combined selective MRQED game.

In the proof, we build a simulator that leverages an MRQED adversary to solve the D-BDH problem or the D-Linear problem. The simulator inherits parameters specified by the D-BDH/D-Linear instance, hence, it has incomplete information about the master key. Therefore, the crux of the proof is how to simulate the key derivation algorithm without knowing the complete master key. In comparison, the anonymity proof is more complicated than the confidentiality proof, because it involves a hybrid argument containing $2DL$ steps. In step (d_1, l_1, n_1) of the hybrid argument, y_{φ_1,n_1} and y'_{φ_1,n_1} $(\varphi_1 = (d_1, l_1))$ in the master key contain unknown parameters inherited from the D-Linear instance. Therefore, we need to condition on the relative position between $\mathbf{X}^*$ and the (d_1, l_1) in question. Our proof techniques are similar to that presented in the AHIBE paper [13].

3.3.5 Practical Performance

In this section, we give a detailed analysis of the performance of the MRQED^D scheme given in Section 3.3.3 in practical scenarios. We use the conditional release of encrypted network audit logs as our motivating application.

Assumptions. To evaluate the scheme of Section 3.3.3 in this application, we detail a set of scenarios regarding the searchable fields present in the logs. We assume log entries contain the fields listed in Table 3.2. The 17-bit time field is sufficient to distinguish times over a period of about 15 years with a one hour resolution, or about three months at a one minute resolution. More precise times may be stored in the non-searchable portion of the message if desired. The protocol

Field	Abbr.	Range	Distinct Values
Source IP	sip	$[0, T_{\mathsf{sip}} - 1]$	$T_{\mathsf{sip}} = 2^{32}$
Dest. IP	dip	$[0, T_{\mathsf{dip}} - 1]$	$T_{\mathsf{dip}} = 2^{32}$
Port	port	$[0, T_{\mathsf{port}} - 1]$	$T_{\mathsf{port}} = 2^{16}$
Time	time	$[0, T_{\mathsf{time}} - 1]$	$T_{\mathsf{time}} = 2^{17}$
Protocol	prot	$[0, T_{\mathsf{prot}} - 1]$	$T_{\mathsf{prot}} = 2^{8}$

Table 3.2: Fields appearing in a network audit log and their possible values.

field corresponds to the actual bits of the corresponding field in an IP header (where, for example, 6 denotes TCP and 133 denotes Fibre Channel). Various subsets of these fields may be included as searchable attributes in MRQED^D. Other fields and any additional associated data such as a payload may be included as the encrypted message. Regardless of message length, we need only use the MRQED^D scheme to encrypt a single group element, which may be a randomly generated symmetric key (e.g., for AES) used to encrypt the message.

Benchmarks for the selected pairing were run on a modern workstation. We ran the benchmarks twice: 1) Back in winter 2006, we used a 64-bit, 3.2 Ghz Pentium 4 processor. 2) We ran the benchmark test then again in summer 2008, on a Intel 2.4GHz Core 2 processor[1]. We used the Pairing-Based Cryptography (PBC) library [29], which is in turn based on the GNU Multiple Precision Arithmetic Library (GMP). Note that the benchmarking program uses a single thread. Therefore, for the dual-core processor, only one core was used in the measurement. The relevant results are given in Table 3.3. It is interesting, but not surprising, to observe that the benchmarks improved by a factor of approximately 2 from 2006 to 2008 (for most of the major operations).

Using these benchmark numbers, we now estimate the performance of our encryption scheme under several scenarios for the network audit log application.

Public parameters and master key. The space required to store the public parameters and master key is logarithmic with respect to the number of possible attribute values. Specifically, denote the set of attributes as $A = \{\mathsf{sip}, \mathsf{dip}, \mathsf{port}, \mathsf{time}, \mathsf{prot}\}$. Then for each attribute $a \in A$, define the

[1] Although the new processor has lower clock cycle than the old one, it is more powerful due to improved pipeline structure.

(a) Year 2006: 64bit 3.2GHz Pentium 4

Operation	Time
pairing (no preprocessing)	5.5 ms
pairing (after preprocessing)	2.6 ms
preprocess pairing	5.9 ms
exponentiation in $\mathbb{G}, \widehat{\mathbb{G}}$	6.4 ms
exponentiation in $\mathbb{G}'$	0.6 ms
multiplication in $\mathbb{G}'$	5.1 μs

(b) Year 2008: 2.40GHz Intel Core(TM)2

Operation	Time
pairing (no preprocessing)	2.6 ms
pairing (after preprocessing)	1.1 ms
preprocess pairing	4.7 ms
exponentiation in $\mathbb{G}, \widehat{\mathbb{G}}$	5.3 ms
exponentiation in $\mathbb{G}'$	0.3 ms
multiplication in $\mathbb{G}'$	2.4 μs

Table 3.3: Group arithmetic and pairing performance benchmarks on a modern workstation. The table on the left reflects benchmarks in 2006. The table on the right reflects updated benchmark numbers in 2008.

height of the tree $L_a = \log_2 T_a + 1$. For example, $L_{\mathsf{sip}} = 33$ and $L_{\mathsf{prot}} = 9$. Then the public parameters PK require a total of $8 \sum_{a \in A} L_a = 880$ elements of $\mathbb{G}$ and one element of $\mathbb{G}'$. Assuming 512-bit representations[2] of elements of $\mathbb{G}$ and $\mathbb{G}'$, the total size of PK is 55KB. The master key SK contains the same number of elements, again requiring 55KB of storage. More space efficient pairings than the one used in this estimate are available, but this one was selected for speed of evaluation.

Computation time for $\mathbf{Setup}$ is reasonable, given that it is only run once. Computing the public and private parameters in $\mathbf{Setup}$ requires roughly $16 \sum_{a \in A} L_a$ exponentiations and one pairing. This means roughly 11.3s running time on the old processor in 2006, and 9.3s on the new processor in 2008. Time spent on multiplication in this case is negligible.

Encryption. Saving the group elements of a ciphertext requires $4 \sum_{a \in A} L_a + 2$ group elements, or 28KB. Note that we normally just encrypt a session key, so this is a constant overhead beyond the actual length of the message. Running $\mathbf{Encrypt}$ requires about two exponentiations for each group element, resulting in a time of about 5.6s in 2006, and 4.7s in 2008. While significant, this overhead should be acceptable in most cases in the network audit log example. If audit logs are high volume, the best strategy may be to produce periodic summaries rather than separately encrypting each packet. The searchable attributes of such summaries would reflect the collection of entries they represent, and the full contents of the entries could be included as the encrypted message without incurring additional overhead. In systems containing a cryptographic accelerator chip supporting ECC (such as some routers), much higher performance is possible. For example, the Elliptic Semiconductor CLP-17 could reduce the time of exponentiation from 6.4ms to 30μs [17], resulting in a total encryption time as low as 27ms.

Key derivation and decryption. We now consider decryption keys and the running time of the decryption algorithm, the more interesting aspects of the scheme's computational and storage requirements. The space required to store a decryption key, the time to derive it, and the time to

[2]We consider a type A pairing using the singular curve $y^2 = x^3 + x$ for the groups $\mathbb{G}$ and $\widehat{\mathbb{G}}$ with a base field size of 512-bits. Note that all groups involved have 160-bit group order; the storage requirements arise from the specific representation of elements in the elliptic curves.

(a) Performance in 2006.

Example Query	N_{sip}, N_{dip}, N_{port}, N_{time}, N_{prot}	Pairing Time	Worst-case Mult. Time	Worst-case Dec. Time
sip = 207.44.178.∗, dip = 216.187.103.169, port = 22, time = ∗, prot = TCP	(1, 1, 1, 1, 1)	65ms	$<$ 0.1ms	65ms
sip ∈ [207.44.178.123, 207.44.182.247], dip = ∗, port = 22, time ∈ [5pm 10/31, 9am 11/5], prot ∈ {TCP, UDP, ICMP}	(10, 1, 1, 7, 3)	286ms	1.2ms	287ms
sip ∈ [207.44.178.123, 207.60.177.15], dip ∈ [207.44.178.123, 207.60.177.15], port ∈ [3024, 35792], time ∈ [10/31/2006, 10/31/2020], prot ∈ {TCP, UDP, ICMP}	(20, 20, 15, 17, 3)	0.98s	1.64s	2.62s

(b) Performance in 2008.

Example Query	N_{sip}, N_{dip}, N_{port}, N_{time}, N_{prot}	Pairing Time	Worst-case Mult. Time	Worst-case Dec. Time
sip = 207.44.178.∗, dip = 216.187.103.169, port = 22, time = ∗, prot = TCP	(1, 1, 1, 1, 1)	28ms	$<$ 0.1ms	28ms
sip ∈ [207.44.178.123, 207.44.182.247], dip = ∗, port = 22, time ∈ [5pm 10/31, 9am 11/5], prot ∈ {TCP, UDP, ICMP}	(10, 1, 1, 7, 3)	121ms	0.6ms	122ms
sip ∈ [207.44.178.123, 207.60.177.15], dip ∈ [207.44.178.123, 207.60.177.15], port ∈ [3024, 35792], time ∈ [10/31/2006, 10/31/2020], prot ∈ {TCP, UDP, ICMP}	(20, 20, 15, 17, 3)	0.41s	0.77s	1.18s

Table 3.4: Decryption times (in 2006 and 2008) resulting from decryption keys of various sizes.

decrypt using it depend only on the ranges of attributes for which it permits decryption. Unlike the computational and storage requirements discussed thus far, these costs do not depend on the full range of possible values, only those associated with the key. These costs depend on the number of key components necessary to represent the permissible range along each dimension. For example, suppose a particular decryption key DK only allows decryption of entries with a destination port in the range $[3, 7]$ (perhaps placing other requirements on the other attributes). Referring back to Figure 3.1, we see that three tree nodes are necessary to cover this range, so $\mathbf{DeriveKey}$ would include these three for the destination port dimension in DK. Similarly, given some decryption key DK, we denote the number of tree nodes necessary to cover the decryption range in each of the dimensions $a \in A$ by $N_a = |\Lambda_a(\mathbf{B})|$ (using the notation of Section 3.3.3). So in this example, $N_{\mathsf{port}} = 3$. Note that for any $a \in A$, in the worst case, $N_a = 2L_a - 2$.

Now given N_a for each $a \in A$, we may compute the decryption costs. A decryption key consists of $5\sum_{a\in A} N_a$ group elements and $\mathbf{DeriveKey}$ performs $8\sum_{a\in A} N_a$ exponentiations. The number of operations necessary to decrypt using a key DK is slightly more subtle. While $\mathbf{QueryDecrypt}$ is $\Theta(\prod_{a\in A} L_a)$ (i.e., $\Theta((\log T)^D)$) overall, only $O(\sum_{a\in A} L_a)$ (i.e., $O(D \log T)$) pairings are required, as mentioned in Section 3.3.3. Specifically, we only need to compute $5\sum_{a\in A} N_a$ pairings to populate a lookup table containing values of $\mathbf{e}(c_0, \mathsf{k}_{ID,0})$, $\mathbf{e}(c_{\varphi,1}, \mathsf{k}_{ID,1})$, $\mathbf{e}(c_{\varphi,2}, \mathsf{k}_{ID,2})$, $\mathbf{e}(c_{\varphi,3}, \mathsf{k}_{ID,3})$, $\mathbf{e}(c_{\varphi,4}, \mathsf{k}_{ID,4})$, and $\mathbf{e}(c_{\varphi,5}, \mathsf{k}_{ID,5})$. These values are sufficient to compute the $\mathbf{QueryDecrypt}$ algorithm. Assuming a key will normally be used to decrypt a batch of ciphertexts one after another, we may further reduce the cost of pairings by preprocessing with the key. As shown in Table 3.3, preprocessing reduces the pairing time by about half, at a one time cost (per decryption key DK) equivalent to one or two decryptions. Computed naively, the sequence of trials in step one of $\mathbf{QueryDecrypt}$ end up requiring a total of $|A| \prod_{a\in A} N_a$ multiplications in $\mathbb{G}'$. This can be somewhat reduced. Let $S_1, \ldots S_{|A|}$ be $\{ N_a \,|\, a \in A \}$ sorted in ascending order: $S_1 \leq S_2 \leq \ldots S_{|A|}$. Then by saving intermediate results between trials and ordering the dimensions appropriately, it is possible to complete step one with a total of $S_1 + S_1S_2 + S_1S_2S_3 + \ldots S_1S_2 \cdots S_{|A|}$ multiplications.

Specific scenarios. We have now computed the costs associated with the storage and usage of a decryption key in terms of N_a for $a \in A$, but we have not yet specified N_a. If we assume the range for each attribute is randomly selected (uniformly), then for each $a \in A$, the expected value of N_a is $L_a - 1$. This results in a decryption key size of 33KB and a running time for $\mathbf{DeriveKey}$ of 5.4s in 2006, and 4.5s in 2008. The corresponding worst-case decryption time[3] is 13.1s in 2006, and 6.1s in 2008. Note that this has improved by a factor of 2 over a period of 1.5 years. This still may be a major cost, and likely to be inconvenient if significant quantities of log entries must be decrypted. Fortunately, queries eliciting such long decryption times are not likely to be necessary in practice. In fact, fairly elaborate queries are possible while keeping decryption costs low.

In Table 3.4 we provide several examples that help demonstrate this. The first entry illustrates the fact that specifying a single value, all values, or a range of values falling on power-of-two boundaries (as in the case of an IP subnet) for some attribute a results in $N_a = 1$, reducing decryption time dramatically. In the next example, several attributes are required to be in general ranges, or, in the case of prot, selected from a small set. This results in larger numbers of key components and slightly longer decryption times. Still, the decryption time in this case is far below the time with each range randomly selected. As shown by the third example, larger ranges result in larger values of N_a and, again, somewhat larger, but still relatively low, decryption times. It is interesting to note that the decryption time has improved by a rough factor of 2 over a period of 1.5 years.

Exploiting parallelism to speed up the computation. The performance numbers in Table 3.4 does not exploit any parallelism. In particular, even for the new dual-core CPU, we did not leverage the dual-core feature, because our benchmarking program used a single thread.

[3]In reality, the average decryption time is smaller than this number, since upon a successful decryption, the $\mathbf{QueryDecrypt}$ algorithm exits after trying half of the combinations in expectation and thus performing half the worst-case multiplications.

We would like to note that most of the above computations are easily parallelizable. For example, if one has multiple entries to decrypt, one can easily distribute them across multiple processors. Even when we are decrypting a single entry, we can parallelize the $\mathbf{QueryDecrypt}$ algorithm internally. For example, the $5\sum_{a\in A} N_a$ pairing operations do not have any dependencies, and we can assign them to different processors easily. We can also distribute the simple hyper-rectangles to the multiple processor: each processor will try to decrypt at k of the $(\log T)^D$ hyper-rectangles.

In fact, if we have plenty of processors, we can distribute the computation such that each processor only has to perform one pairing. After all the pairing results are computed, each processor tries to decrypt at one simple rectangle. In this case, the multiplication time becomes negligible compared to the pairing time. Therefore, ignoring possible overheads of parallelism, the theoretic decryption time can be improved to roughly the time of a single pairing operation (1.1ms as of 2008), even in the worst-case scenario.

This is very encouraging, especially as parallel computation is starting to be widely accepted in practice. The latest consumer PCs have multiple processors, and IT companies are using large clusters to run their computations. For example, Google's cluster has an estimate of 100K nodes, and this may well be a conservative estimate [38].

3.4 The Dual Problem and Stock Trading through a Broker

In the MRQED problem, one encrypts a message $\mathbf{Msg}$ under a point $\mathbf{X}$ in multi-dimensional space, and given a hyper-rectangle $\mathbf{B}$, the master key owner can construct a capability, allowing an auditor to decrypt all entries satisfying $\mathbf{X} \in \mathbf{B}$. On the other hand, the privacy of the irrelevant entries are still preserved.

Informally, the natural dual problem to MRQED is where one encrypts under a hyper-rectangle $\mathbf{B}$, and given a point $\mathbf{X}$, the master key owner can construct a capability allowing an auditor to decrypt all entries satisfying $\mathbf{B} \ni \mathbf{X}$. Like in MRQED, we require that the privacy of all irrelevant entries be preserved. We now show an interesting application of the dual problem, and then show that MRQED implies a solution for the dual problem.

An interesting application of the dual problem is for trading stocks and other securities. Suppose an *investor* trades stocks through a *broker*. The investor specifies a price range and a time range, such that if the stock price falls within that range during a specific period of time, the broker can buy or sell the stock on behalf of the investor. This is usually referred to as a *stop order*, *limit order*, or *stop-limit order*. Sometimes, the investor may not fully trust the broker, and may wish to conceal the price and time ranges from the broker before an order is executed.

The dual problem can be applied in such scenarios to address the privacy concerns of investors. In particular, the *stock exchange*, or any third-party with knowledge of the real-time stock price can act as the trusted authority who owns the master key. For convenience, in the following description, we assume that the *stock exchange* is the trusted authority. The investor first encrypts the order along with the desired price and time ranges, and sends the encrypted order to the broker. Suppose that at a certain point of time t, the stock price is p. The stock exchange constructs a decryption key for the pair (t, p), and hands it to the broker. With this decryption key, the broker can decrypt all orders whose price and time ranges match the current price p and the current time t, and execute

these orders. For orders whose price and time ranges do not match the current price and time, the broker cannot learn any additional information about these orders.

MRQED implies the dual problem. We use a 2-dimensional example to illustrate how MRQED implies a solution for the dual problem.

- Dual.$\mathbf{Setup}$ $(\Sigma, [T]^2)$: Call MRQED.$\mathbf{Setup}$ $(\Sigma, [T]^4)$, and output the public key PK, and master key SK.
- Dual.$\mathbf{Encrypt}$ $(\mathsf{PK}, [x_1, x_2] \times [y_1, y_2], \mathbf{Msg})$: To encrypt a message $\mathbf{Msg}$ under the range $[x_1, x_2] \times [y_1, y_2]$ in 2 dimensions, call MRQED.$\mathbf{Encrypt}$ $(\mathsf{PK}, (x_1, x_2, y_1, y_2), \mathbf{Msg})$. Observe that here a range $[x_1, x_2] \times [y_1, y_2]$ in $[T]^2$ is mapped to a point (x_1, x_2, y_1, y_2) in $[T]^4$.
- Dual.$\mathbf{DeriveKey}$ $(\mathsf{PK}, \mathsf{SK}, (x, y))$: To generate a decryption key for the point $(x, y) \in [T]^2$, call MRQED.$\mathbf{DeriveKey}$ $(\mathsf{PK}, \mathsf{SK}, [1, x] \times [x, T] \times [1, y] \times [y, T])$.
- Dual.$\mathbf{QueryDecrypt}$ $(\mathsf{PK}, \mathbf{DK}, \mathbf{C})$: To try to decrypt a ciphertext $\mathbf{C}$ using the decryption key $\mathbf{DK}$, call MRQED.$\mathbf{QueryDecrypt}$ $(\mathsf{PK}, \mathbf{DK}, \mathbf{C})$.

In essence, the above scheme maps a range $[x_1, x_2] \times [y_1, y_2] \subseteq [T]^2$ to a point $(x_1, x_2, y_1, y_2) \in [T]^4$, and testing if a point (x, y) is within the range $[x_1, x_2] \times [y_1, y_2]$ is equivalent to testing whether $(x_1, x_2, y_1, y_2) \in [1, x] \times [x, T] \times [1, y] \times [y, T]$. It is easy to verify that the security of the MRQED scheme guarantees a similar notion of security for the dual construction, i.e., if a decryption key fails to decrypt a certain ciphertext entry, then a probabilistic polynomial adversary cannot learn any additional information about that entry.

3.5 Notation

We summarize the notations used throughout this chapter in Table 3.5.

Notation	Explanation	First Defined
$[s,t]$	integers s through t	Sec. 3.1.2
$[a]$	integers 1 through a	Sec. 3.1.2
D	number of dimensions	Sec. 3.1.2
T	number of discrete values in each dimension	Sec. 3.1.2
$\mathbb{L}_{\Delta}$	multi-dimensional lattice	Sec. 3.1.2
$\mathbf{X}$	a point in the lattice	Sec. 3.1.2
$\mathbf{B}$	a hyper-rectangle	Sec. 3.1.2
Σ	security parameter	Sec. 3.1.2
PK	public key	Sec. 3.1.2
SK	master key	Sec. 3.1.2
$\mathbf{DK}$	decryption key	Sec. 3.1.2
$\mathbf{Msg}$	message to encrypt	Sec. 3.1.2
$\mathbb{M}$	message space	Sec. 3.1.2
$\mathbf{G}$	a bilinear instance	Sec. 3.3.1
$\mathbb{G}$	bilinear group	Sec. 3.3.1
$\mathbb{G}'$	target group	Sec. 3.3.1
e	bilinear pairing function	Sec. 3.3.1
g	generator of $\mathbb{G}$	Sec. 3.3.1
$\mathbb{Z}_p$	additive group of integers modular a prime p	Sec. 3.3.1
$\mathbb{Z}_p^*$	multiplicative group of integers modular a prime p	Sec. 3.3.3
$\mathsf{tr}(T)$	binary interval tree over integers 1 through T	Sec. 3.2.2
ID	identity of a tree node	Sec. 3.2.2
$\mathsf{cv}(ID)$	range represented by a tree node ID	Sec. 3.2.2
$\mathbb{P}(x)$	path from the root to the leaf node representing x	Sec. 3.2.2
$\Lambda(s,t)$	set of nodes representing the range [s, t]	Sec. 3.2.2
$\Lambda_d(\mathbf{B})$	set of nodes representing the range specified by $\mathbf{B}$ in the d^{th} dimension	Sec. 3.2.3
$\mathbf{B}_0$	simple hyper-rectangle	Sec. 3.2.3
$\mathsf{id}_{\mathbf{B}_0}$	identity vector of the simple hyper-rectangle $\mathbf{B}_0$	Sec. 3.2.3
$\Lambda^{\times}(\mathbf{B})$	hyper-rectangle $\mathbf{B}$ as a collection of simple hyper-rectangles	Sec. 3.2.3
$\mathbb{P}_d(\mathbf{X})$	path to root in the d^{th} dimension for the point $\mathbf{X}$	Sec. 3.2.3
$\mathbb{P}^{\times}(\mathbf{X})$	cross-product of all D paths to root for the point $\mathbf{X}$	Sec. 3.2.3
$\mathbb{P}^{\cup}(\mathbf{X})$	union of all D paths to root for the point $\mathbf{X}$	Sec. 3.3.2
$\Lambda^{\cup}(\mathbf{B})$	hyper-rectangle $\mathbf{B}$ as a set of tree nodes	Sec. 3.3.2
L	height of interval tree	Sec. 3.3.3
$\Phi(ID)$	a function that outputs the dimension and depth of some node ID	Sec. 3.3.3
$\varphi = (d,l)$	usually used in subscripts to indicate the dimension and depth respectively	Sec. 3.3.3
$\mathcal{I}_{\varphi}(\mathbf{X})$ where $\varphi = (d,l)$	the node at depth l in the path $\mathbb{P}_d(\mathbf{X})$ of the d^{th} dimension	Sec. 3.3.3

Table 3.5: Notations.

3.6 Proof of Consistency

Proof of Theorem 3.3.2:

Let $\mathbf{C} = \left(c, c_0, \left[c_{\varphi,1}, c_{\varphi,2}, c_{\varphi,3}, c_{\varphi,4}\right]_{\varphi=(d,l)\in[D]\times[L]}\right)$ be the encryption of $\mathbf{Msg}$ on point $\mathbf{X}$. Let $\Lambda^{\times}(\mathbf{B}_0) = \{(ID_1, ID_2, \ldots, ID_D)\} \subseteq \Lambda^{\times}(\mathbf{B})$ be the current simple hyper-rectangle under decryption. Let $\varphi_d = \Phi(ID_d)$ ($d \in [D]$).

If $\mathbf{X} \in \mathbf{B}_0$, then for all $d \in [D]$, $\mathcal{I}_{\varphi_d}(\mathbf{X}) = ID_d$. For simplicity, let $\xi(x) = \mathbf{e}(g,g)^x$, and denote

$\mathcal{I}_{\varphi_d} = \mathcal{I}_{\varphi_d}(\mathbf{X})$. Now decryption for $\mathbf{B_0}$ proceeds as follows:

$$
\begin{aligned}
V =&(\mathbf{Msg}||0^{m'}) \cdot \Omega^{-r} \cdot \prod_{d\in[D]} \mathbf{e}\left(g^r, \widetilde{\mu}_d \left(y_{\varphi_d,1}{}^{ID_d} y'_{\varphi_d,1}\right)^{\lambda_{ID_d,1}} \left(y_{\varphi_d,2}{}^{ID_d} y'_{\varphi_d,2}\right)^{\lambda_{ID_d,2}}\right) \\
&\cdot \prod_{d\in[D],n\in[2]} \mathbf{e}\left(\mathsf{a}_{\varphi_d,n}{}^{-\lambda_{ID_d,n}}, \left(b_{\varphi_d,n}{}^{\mathcal{I}_{\varphi_d}} b'_{\varphi_d,n}\right)^{r_{\varphi_d,n}}\right) \cdot \prod_{d\in[D],n\in[2]} \mathbf{e}\left(\mathsf{b}_{\varphi_d,n}{}^{-\lambda_{ID_d,n}}, \left(a_{\varphi_d,n}{}^{\mathcal{I}_{\varphi_d}} a'_{\varphi_d,n}\right)^{r-r_{\varphi_d,n}}\right) \\
=&(\mathbf{Msg}||0^{m'}) \cdot \Omega^{-r} \cdot \mathbf{e}\left(g^r, \widetilde{\omega}\right) \cdot \xi\left(r \cdot \sum_{\substack{d\in[D],\\ n\in[2]}} \alpha_{\varphi_d,n}\beta_{\varphi_d,n}\lambda_{ID_d,n}\left(\theta_{\varphi_d,n} ID_d + \theta'_{\varphi_d,n}\right)\right) \\
&\cdot \xi\left(\sum_{\substack{d\in[D],\\ n\in[2]}} \alpha_{\varphi_d,n}(-\lambda_{ID_d,n}) r_{\varphi_d,n}\beta_{\varphi_d,n}\left(\theta_{\varphi_d,n}\mathcal{I}_{\varphi_d} + \theta'_{\varphi_d,n}\right)\right) \\
&\cdot \xi\left(\sum_{\substack{d\in[D],\\ n\in[2]}} \beta_{\varphi_d,n}(-\lambda_{ID_d,n})\left(r - r_{\varphi_d,n}\right)\alpha_{\varphi_d,n}\left(\theta_{\varphi_d,n}\mathcal{I}_{\varphi_d} + \theta'_{\varphi_d,n}\right)\right) \\
=&(\mathbf{Msg}||0^{m'}) \cdot \Omega^{-r} \cdot \mathbf{e}\left(g^r, \widetilde{\omega}\right) \cdot \xi\left(r \cdot \sum_{\substack{d\in[D],\\ n\in[2]}} \alpha_{\varphi_d,n}\beta_{\varphi_d,n}\lambda_{ID_d,n}\left(\theta_{\varphi_d,n} ID_d + \theta'_{\varphi_d,n}\right)\right) \\
&\cdot \xi\left(r \cdot \sum_{\substack{d\in[D],\\ n\in[2]}} \alpha_{\varphi_d,n}\beta_{\varphi_d,n}(-\lambda_{ID_d,n})\left(\theta_{\varphi_d,n}\mathcal{I}_{\varphi_d} + \theta'_{\varphi_d,n}\right)\right) \\
=&\mathbf{Msg}||0^{m'}.
\end{aligned}
$$

Else if $\mathbf{X} \notin \mathbf{B}_0$, $\mathcal{I}_{\varphi_d}(\mathbf{X}) \neq ID_d$, $d \in [D]$. Hence decryption yields

$$
\begin{aligned}
V &= (\mathbf{Msg}||0^{m'}) \cdot \frac{\xi\left(r \cdot \sum_{\substack{d\in[D],\\ n\in[2]}} \alpha_{\varphi_d,n}\beta_{\varphi_d,n}\lambda_{ID_d,n}\left(\theta_{\varphi_d,n} ID_d + \theta'_{\varphi_d,n}\right)\right)}{\xi\left(r \cdot \sum_{\substack{d\in[D],\\ n\in[2]}} \alpha_{\varphi_d,n}\beta_{\varphi_d,n}\lambda_{ID_d,n}\left(\theta_{\varphi_d,n}\mathcal{I}_{\varphi_d} + \theta'_{\varphi_d,n}\right)\right)} \\
&= (\mathbf{Msg}||0^{m'}) \cdot Q^r
\end{aligned}
$$

where

$$Q = \xi \left(\sum_{\substack{d \in [D], \\ n \in [2]}} \alpha_{\varphi_d,n} \beta_{\varphi_d,n} \lambda_{ID_d,n} \left(\theta_{\varphi_d,n} ID_d + \theta'_{\varphi_d,n} \right) - \sum_{\substack{d \in [D], \\ n \in [2]}} \alpha_{\varphi_d,n} \beta_{\varphi_d,n} \lambda_{ID_d,n} \left(\theta_{\varphi_d,n} \mathcal{I}_{\varphi_d} + \theta'_{\varphi_d,n} \right) \right)$$

With probability $1 - 1/p$, $Q \neq 1$, and the ciphertext is distributed uniformly at random in $\mathbb{G}'$. Hence the probability that V is of the form $\widehat{\mathbf{Msg}}||0^{m'}$ is less than $\frac{1}{p} + \frac{1}{2^{m'}}$.

3.7 Proof of Security

To prove the selective security of our MRQED^D construction, we decompose the selective MRQED game into two games: a selective confidentiality game and a selective anonymity game. By the hybrid argument, if no polynomial-time adversary has more than negligible advantage in either the confidentiality game or the anonymity game, then no polynomial-time adversary has more than negligible advantage in the combined selective MRQED game. The terminology *confidentiality* and *anonymity* that we use here is adopted from AIBE schemes.

Definition 3.7.1 (MRQED selective confidentiality game) *The MRQED selective confidentiality game is defined as below.*

- **Init**: The adversary $\mathcal{A}$ outputs a point $\mathbf{X}^*$ where it wishes to be challenged.
- **Setup**: The challenger runs the $\mathbf{Setup}(\mathbf{\Sigma}, \mathbb{L}_{\mathbf{\Delta}})$ algorithm to generate PK, SK. It gives PK to the adversary, but does not divulge SK.
- **Phase 1**: The adversary is allowed to issue decryption key queries for hyper-rectangles that do not contain $\mathbf{X}^*$.
- **Challenge**: The adversary submits two equal length messages $\mathbf{Msg}_0$ and $\mathbf{Msg}_1$. The challenger flips a random coin, b, and encrypts $\mathbf{Msg}_b$ under $\mathbf{X}^*$. The ciphertext is passed to the adversary.
- **Phase 2**: Phase 1 is repeated.
- **Guess**: The adversary outputs a guess b' of b.

Definition 3.7.2 (MRQED selective anonymity game) *The MRQED selective anonymity game is defined as below.*

- **Init**: The adversary $\mathcal{A}$ outputs two points $\mathbf{X}_0$ and $\mathbf{X}_1$, where it wishes to be challenged.
- **Setup**: The challenger runs the $\mathbf{Setup}(\mathbf{\Sigma}, \mathbb{L}_{\mathbf{\Delta}})$ algorithm to generate PK, SK. It gives PK to the adversary, but does not divulge SK.
- **Phase 1**: The adversary is allowed to issue decryption key queries for hyper-rectangles that do not contain $\mathbf{X}_0$ and $\mathbf{X}_1$.
- **Challenge**: The adversary submits a message $\mathbf{Msg}$. The challenger first flips a random coin b, and then encrypts $\mathbf{Msg}$ under $\mathbf{X}_b$. The ciphertext is passed to the adversary.

- **Phase 2**: Phase 1 is repeated.
- **Guess**: The adversary outputs a guess b' of b.

In either game, we define the adversary $\mathcal{A}$'s advantage as

$$\mathbf{Adv}_{\mathcal{A}}(\Sigma) = \left| \Pr[b = b'] - \frac{1}{2} \right|$$

Definition 3.7.3 (IND-sID-CPA) *An MRQED scheme is IND-sID-CPA secure if all polynomial-time adversaries have at most a negligible advantage in the confidentiality game.*

Definition 3.7.4 (ANON-sID-CPA) *An MRQED scheme is ANON-sID-CPA secure if all polynomial-time adversaries have at most a negligible advantage in the anonymity game.*

Lemma 3.7.5 *If an MRQED scheme is both IND-sID-CPA secure and ANON-sID-CPA secure, then the MRQED scheme is selectively secure.*

Proof: By the hybrid argument. ■

Hence, it suffices to prove our MRQED construction IND-sID-CPA and ANON-sID-CPA secure. We say that an MRQED scheme is (τ, q, ϵ) secure if any adversary making q range queries for decryption keys, cannot have more than ϵ advantage within time τ.

Theorem 3.7.6 (Confidentiality) *Suppose $\mathbf{G}$ satisfies the (τ, ϵ) D-BDH assumption, then the above defined MRQED scheme is (τ', q, ϵ) IND-sID-CPA secure, where $\tau' < \tau - \Theta(qD \log T)$.*

Theorem 3.7.7 (Anonymity) *Suppose $\mathbf{G}$ satisfies the (τ, ϵ) D-Linear assumption, then the above defined MRQED scheme is (τ', q, ϵ') ANON-sID-CPA secure, where $\tau' < \tau - \Theta(qD \log T)$, and $\epsilon' = (2D \log T + 1)(\epsilon + 1/p)$.*

In particular, $\Theta(qD \log T)$ comes from the fact that the simulator needs $O(D \log T)$ time to compute the decryption key for each hyper-rectangle queried. The $2D \log T + 1$ loss factor in ϵ' comes from the hybrid argument we use to prove anonymity, and additive $1/p$ comes from the probability that bad events happen in the simulation so that the simulator has to abort.

■

3.7.1 Proof: Confidentiality

Proof of Theorem 3.7.6:

We reduce the semantic security of MRQED to the hardness of the D-BDH problem. Let $[g, g_1, g_2, g_3, Z]$ denote the D-BDH instance supplied to the simulator, $\mathcal{B}$, where $g_1 = g^{z_1}$, $g_2 = g^{z_2}$, $g_3 = g^{z_3}$, the simulator's task is to decide whether or not $Z = \mathbf{e}(g, g)^{z_1 z_2 z_3}$. And to do this, the simulator leverages an MRQED IND-sID-CPA adversary, $\mathcal{A}$.

We describe a reduction such that if $Z = \mathbf{e}(g, g)^{z_1 z_2 z_3}$, the simulator produces a valid ciphertext; otherwise, the first term c in the ciphertext is random. Hence, if the adversary could break the confidentiality of the scheme, the simulator would be able to solve the D-BDH problem.

Init: The adversary selects a point $\mathbf{X}^* \in \mathbb{L}_\Delta$ that it wishes to attack. For $\varphi \in [D] \times [L]$, define $\mathcal{I}^*_\varphi = \mathcal{I}_\varphi(\mathbf{X}^*)$.

Setup: To create public and private parameters, the simulator does the following:

1. Pick at random from ${\mathbb{Z}_p}^{12DL}$:

$$\left[\alpha_{\varphi,n}, \beta_{\varphi,n}, \theta_{\varphi,n}, \theta'_{\varphi,n}, \bar{\theta}_{\varphi,n}, \bar{\theta}'_{\varphi,n}\right]_{\varphi=(d,l)\in[D]\times[L], n\in[2]}$$

subject to the constraint that

$$\left[\bar{\theta}_{\varphi,n}\mathcal{I}^*_\varphi + \bar{\theta}'_{\varphi,n} = 0\right]_{\varphi=(d,l)\in[D]\times[L], n\in[2]}$$

where $\mathcal{I}^*_\varphi = \mathcal{I}_\varphi(\mathbf{X}^*)$. We also require that the α's, β's, $\bar{\theta}$'s and $\bar{\theta}'$'s are forcibly non-zero.

2. Release the following public parameters to the adversary.

$$\Omega \leftarrow \mathbf{e}(g_1, g_2), \left[\begin{array}{ll} a_{\varphi,n} \leftarrow \left(g^{\theta_{\varphi,n}} {g_1}^{\bar{\theta}_{\varphi,n}}\right)^{\alpha_{\varphi,n}}, & a'_{\varphi,n} \leftarrow \left(g^{\theta'_{\varphi,n}} {g_1}^{\bar{\theta}'_{\varphi,n}}\right)^{\alpha_{\varphi,n}}, \\ b_{\varphi,n} \leftarrow \left(g^{\theta_{\varphi,n}} {g_1}^{\bar{\theta}_{\varphi,n}}\right)^{\beta_{\varphi,n}}, & b'_{\varphi,n} \leftarrow \left(g^{\theta'_{\varphi,n}} {g_1}^{\bar{\theta}'_{\varphi,n}}\right)^{\beta_{\varphi,n}}, \end{array}\right]_{\varphi=(d,l)\in[D]\times[L], n\in[2]}$$

Note that this posits that $\omega = z_1 z_2$; in addition, both ω and $\widetilde{\omega}$ are both unknown to the simulator.

3. Compute what it can of the master key.

$$\left[\begin{array}{ll} \mathsf{a}_{\varphi,n} \leftarrow g^{\alpha_{\varphi,n}}, & \mathsf{b}_{\varphi,n} \leftarrow g^{\beta_{\varphi,n}}, \\ y_{\varphi,n} \leftarrow \left(g^{\theta_{\varphi,n}} {g_1}^{\bar{\theta}_{\varphi,n}}\right)^{\alpha_{\varphi,n}\beta_{\varphi,n}}, & y'_{\varphi,n} \leftarrow \left(g^{\theta'_{\varphi,n}} {g_1}^{\bar{\theta}'_{\varphi,n}}\right)^{\alpha_{\varphi,n}\beta_{\varphi,n}} \end{array}\right]_{\varphi=(d,l)\in[D]\times[L], n\in[2]}$$

Portion $\widetilde{\omega}$ of the master key is unknown to the simulator.

Phase 1: Suppose the adversary makes a decryption key query for the hyper-rectangle

$$\mathbf{B}(s_1, t_1, s_2, t_2, \ldots, s_D, t_D)$$

Since $\mathbf{B}$ does not contain $\mathbf{X}^*$, there exists a dimension $d_0 \in [D]$ such that $x^*_{d_0} \notin [s_{d_0}, t_{d_0}]$, where $x^*_{d_0}$ is $\mathbf{X}^*$ projected onto the d_0^{th} dimension. Hence, there exists a dimension $d_0 \in [D]$, such that for all $ID \in \Lambda_{d_0}(\mathbf{B})$, $ID \neq \mathcal{I}^*_\varphi$, where $\varphi = (d_0, l) = \Phi(ID)$. We say that $\mathbf{X}^*$ does not overlap with $\mathbf{B}$ in dimension d_0. The simulator now does the following:

1. Pick d_0 such that $\mathbf{X}^*$ does not overlap with $\mathbf{B}$ in dimension d_0. Let $n_0 = 1$.
2. Pick the following numbers at random from $\mathbb{Z}_p^{D+2|\Lambda^\cup(\mathbf{B})|}$:

$$\left[\mu_d\right]_{d\in[D]}, \left[\widetilde{\lambda}_{ID,n_0}\right]_{ID\in\Lambda_{d_0}(\mathbf{B})}, \left[\lambda_{ID,n}\right]_{ID\in\Lambda_{d_0}(\mathbf{B}), n\neq n_0}, \left[\lambda_{ID,n}\right]_{ID\in\Lambda^\cup(\mathbf{B})-\Lambda_{d_0}(\mathbf{B}), n\in[2]}$$

subject to the constraint that $\sum_{d\in[D]} \mu_d = 0$.

3. For all $ID \in \Lambda^\cup(\mathbf{B}) - \Lambda_{d_0}(\mathbf{B})$, let $\mathbf{DK}(ID) = \left(\mathsf{k}_{ID,0}, \left[\mathsf{k}^{(\mathsf{a})}_{ID,1}, \mathsf{k}^{(\mathsf{b})}_{ID,1}\right], \left[\mathsf{k}^{(\mathsf{a})}_{ID,2}, \mathsf{k}^{(\mathsf{b})}_{ID,2}\right]\right)$ represent the element in $\mathbf{DK}$ for ID, let $\varphi = (d, l) = \Phi(ID)$ where $d \neq d_0$, compute and release $\mathbf{DK}(ID)$ as below:

$$\mathsf{k}_{ID,0} \leftarrow g^{\mu_d} \cdot \prod_{n\in[2]} \left({y_{\varphi,n}}^{ID} y'_{\varphi,n}\right)^{\lambda_{ID,n}},$$
$$\left[\mathsf{k}^{(\mathsf{a})}_{ID,n} \leftarrow {\mathsf{a}_{\varphi,n}}^{-\lambda_{ID,n}}, \mathsf{k}^{(\mathsf{b})}_{ID,n} \leftarrow {\mathsf{b}_{\varphi,n}}^{-\lambda_{ID,n}}\right]_{n\in[2]}$$

4. For all $ID \in \Lambda_{d_0}(\mathbf{B})$, let $\varphi_0 = (d_0, l) = \Phi(ID)$, compute and release $\mathbf{DK}(ID)$ as below:

$$\mathsf{k}_{ID,0} \leftarrow \widetilde{\omega} g^{\mu_{d_0}} \cdot \prod_{n\in[2]} \left(y_{\varphi_0,n}{}^{ID} y'_{\varphi_0,n}\right)^{\lambda_{ID,n}},$$
$$\left[\mathsf{k}^{(\mathsf{a})}_{ID,n} \leftarrow \mathsf{a}_{\varphi_0,n}{}^{-\lambda_{ID,n}}, \mathsf{k}^{(\mathsf{b})}_{ID,n} \leftarrow \mathsf{b}_{\varphi_0,n}{}^{-\lambda_{ID,n}}\right]_{n\in[2]}$$

where

$$\lambda_{ID,n_0} = \widetilde{\lambda}_{ID,n_0} - \frac{z_2}{\alpha_{\varphi_0,n_0}\beta_{\varphi_0,n_0}\bar{\Theta}_{\varphi_0,n_0}} \tag{3.2}$$
$$\bar{\Theta}_{\varphi_0,n_0} = \bar{\theta}_{\varphi_0,n_0} ID + \bar{\theta}'_{\varphi_0,n_0} \neq 0.$$

This ensures that λ_{ID,n_0} is distributed uniformly at random in $\mathbb{Z}_p$. Since $\bar{\theta}_{\varphi_0,n_0}\mathcal{I}^*_{\varphi_0} + \bar{\theta}'_{\varphi_0,n_0} = 0$; moreover, the simulator has picked d_0 such that $ID \neq \mathcal{I}^*_{\varphi_0}$, we then have $\bar{\Theta}_{\varphi_0,n_0} \neq 0$. Although the simulator does not know λ_{ID,n_0} (since it does not know z_2), it can compute $\mathsf{a}_{\varphi_0,n_0}{}^{-\lambda_{ID,n_0}}$ and $\mathsf{b}_{\varphi_0,n_0}{}^{-\lambda_{ID,n_0}}$ given g^{z_2}. Since the simulator does not know $\widetilde{\omega}$, we now explain how to compute $\mathsf{k}_{ID,0}$. The simulator rewrites the equation for $\mathsf{k}_{ID,0}$ as

$$\mathsf{k}_{ID,0} = \left[g^{\mu_{d_0}} \cdot \left(y_{\varphi_0,2}{}^{ID} y'_{\varphi_0,2}\right)^{\lambda_{ID,2}}\right] \cdot \widetilde{\omega} \cdot \left(y_{\varphi_0,1}{}^{ID} y'_{\varphi_0,1}\right)^{\lambda_{ID,1}}$$

Let $\Psi = g^{\mu_{d_0}} \cdot \left(y_{\varphi_0,2}{}^{ID} y'_{\varphi_0,2}\right)^{\lambda_{ID,2}}$, then $\mathsf{k}_{ID,0} = \Psi \cdot \widetilde{\omega} \cdot \left(y_{\varphi_0,n_0}{}^{ID} y'_{\varphi_0,n_0}\right)^{\lambda_{ID,n_0}}$ The simulator can compute part Ψ because it possesses all necessary parameters required to compute it.

Although the simulator cannot directly compute the value of λ_{ID,n_0} (since it does not know z_2), it is capable of computing $\mathsf{k}_{ID,0}$ given g^{z_1} and g^{z_2}; since if we rewrite $\mathsf{k}_{ID,0}$ as below, we can see that the exponent only contains z_1 and z_2 to the first degree. For convenience, we omit the subscripts φ_0, n_0 and ID below by letting $\alpha = \alpha_{\varphi_0,n_0}$, $\beta = \beta_{\varphi_0,n_0}$, $\theta = \theta_{\varphi_0,n_0}$, $\theta' = \theta'_{\varphi_0,n_0}$, $\bar{\theta} = \bar{\theta}_{\varphi_0,n_0}$, $\bar{\theta}' = \bar{\theta}'_{\varphi_0,n_0}$, $y = y_{\varphi_0,n_0}$, $y' = y'_{\varphi_0,n_0}$, $\overline{\Theta} = \overline{\Theta}_{\varphi_0,n_0}$, $\lambda = \lambda_{ID,n_0}$, $\widetilde{\lambda} = \widetilde{\lambda}_{ID,n_0}$.

$$\begin{aligned}\mathsf{k}_{ID,0} =& \Psi \cdot g^{z_1 z_2} \cdot \left(y^{ID} y'\right)^{\lambda} = \Psi \cdot g^{z_1 z_2} \cdot \left(g^{\alpha\beta(\theta + z_1\bar{\theta})ID} g^{\alpha\beta(\theta' + z_1\bar{\theta}')}\right)^{\widetilde{\lambda} - z_2/(\alpha\beta\overline{\Theta})} \\ =& \Psi \cdot g^{z_1 z_2} \cdot g^{-z_1 z_2(\bar{\theta}\cdot ID + \bar{\theta}')/\overline{\Theta}} \cdot g^{\mathsf{f}(z_1,z_2,\alpha,\beta,\theta,\theta',\bar{\theta},\bar{\theta}',\widetilde{\lambda},\overline{\Theta},ID)} = \Psi \cdot g^{\mathsf{f}(z_1,z_2,\alpha,\beta,\theta,\theta',\bar{\theta},\bar{\theta}',\widetilde{\lambda},\overline{\Theta},ID)}\end{aligned}$$

where $\mathsf{f}(z_1, z_2, \alpha, \beta, \theta, \theta', \bar{\theta}, \bar{\theta}', \widetilde{\lambda}, \overline{\Theta}, ID)$ is a polynomial where variables z_1 and z_2 have maximum degree 1.

Challenge: The adversary gives the simulator two messages $\mathbf{Msg}_0$ and $\mathbf{Msg}_1$. The simulator picks a random bit b, and encrypts $\mathbf{Msg}_b$ under point $\mathbf{X}^*$ as below:

1. Pick random integers $[r_{\varphi,n}]_{\varphi=(d,l)\in[D]\times[L],n\in[2]} \in \mathbb{Z}_p^{2DL}$.
2. Compute and release the following as the ciphertext.

$$(\mathbf{Msg}_b || 0^{m'}) \cdot Z^{-1}, g_3, \left[g^{r_{\varphi,n}\beta_{\varphi,n}\left(\theta_{\varphi,n}\mathcal{I}^*_{\varphi} + \theta'_{\varphi,n}\right)}, \left(g_3 \cdot g^{-r_{\varphi,n}}\right)^{\alpha_{\varphi,n}\left(\theta_{\varphi,n}\mathcal{I}^*_{\varphi} + \theta'_{\varphi,n}\right)}\right]_{\varphi=(d,l)\in[D]\times[L],n\in[2]}$$

Note that this implies that $r = z_3$; and if $Z = \mathbf{e}(g,g)^{z_1 z_2 z_3}$, it is easy to verify that the ciphertext is well-formed, due to the fact that $\left[\bar{\theta}_{\varphi,n}\mathcal{I}^*_\varphi + \bar{\theta}'_{\varphi,n} = 0\right]_{\varphi=(d,l)\in[D]\times[L],n\in[2]}$. On the other hand, if Z is a random number, then the first term c in the ciphertext is random and independent of the remaining terms.

Phase 2: Phase 1 is repeated.

Guess: When the adversary outputs a guess b' of b, the simulator outputs 1 if $b' = b$ and 0 otherwise, in answer to the D-BDH instance. ■

3.7.2 Proof: Anonymity

In Definition 3.7.2 of the selective-ID anonymity game, the challenger flips a random coin b in the **Challenge** phase. An equivalent definition is where the challenger flips the coin b in the **Setup** phase before running the $\mathbf{Setup}(\mathbf{\Sigma}, \mathbb{L}_{\mathbf{\Delta}})$ algorithm. This new definition can be further translated into a real-or-random version which we will use in the following proof of anonymity. In the real-or-random game, the adversary commits to only one point $\mathbf{X}^*$ in the **Init** phase; any of its subsequent range queries must not contain $\mathbf{X}^*$; in the **Challenge** phase, the challenger either returns a faithful encryption of $\mathbf{Msg}$ under $\mathbf{X}^*$ or a completely random ciphertext; and the adversary's job is to distinguish between these two worlds. It is easy to verify that the above real-or-random definition implies the selective-ID anonymity definition as stated in Definition 3.7.2 [13].

The proof of anonymity is carried out in $2DL$ steps using a hybrid argument. To do this, we define the following games, where $*$ represents a number distributed uniformly at random from the appropriate group.

$$
\begin{aligned}
\mathbb{W}_{real}: &\quad \text{The challenge ciphertext is } \left(c, c_0, [c^{(b)}_{(1,1),1}, c^{(a)}_{(1,1),1}], \ldots, [c^{(b)}_{(D,L),2}, c^{(a)}_{(D,L),2}]\right);\\
\mathbb{W}_0: &\quad \text{The challenge ciphertext is } \left(*, c_0, [c^{(b)}_{(1,1),1}, c^{(a)}_{(1,1),1}], \ldots, [c^{(b)}_{(D,L),2}, c^{(a)}_{(D,L),2}]\right);\\
\mathbb{W}_{1,1,1}: &\quad \text{The challenge ciphertext is } \left(*, c_0, [*,*], [c^{(b)}_{(1,1),2}, c^{(a)}_{(1,1),2}], \ldots, [c^{(b)}_{(D,L),2}, c^{(a)}_{(D,L),2}]\right);\\
\mathbb{W}_{1,1,2}: &\quad \text{The challenge ciphertext is } \left(*, c_0, [*,*], [*,*], [c^{(b)}_{(1,2),1}, c^{(a)}_{(1,2),1}], \ldots, [c^{(b)}_{(D,L),2}, c^{(a)}_{(D,L),2}]\right);\\
&\ldots\\
\mathbb{W}_{D,L,1}: &\quad \text{The challenge ciphertext is } \left(*, c_0, [*,*], [*,*], \ldots, [*,*], [c^{(b)}_{(D,L),2}, c^{(a)}_{(D,L),2}]\right);\\
\mathbb{W}_{D,L,2}: &\quad \text{The challenge ciphertext is } \left(*, c_0, [*,*], [*,*], \ldots, [*,*], [*,*]\right).
\end{aligned}
$$

In step (d,l,n) of the hybrid argument, we show that $\mathbb{W}_{d,l,n}$ is computationally indistinguishable from the previous world. Note that the transition from $\mathbb{W}_{real}$ to $\mathbb{W}_0$ is the standard concept of semantic security, and has been proved in the previous section. In addition, $\mathbb{W}_{D,L,2}$ is computationally indistinguishable from a completely random ciphertext, hence is anonymous.

We reduce the anonymity of our MRQED scheme to the hardness of the D-Linear problem. We rewrite the D-Linear problem as given $[g, g^{z_1}, g^{z_2}, Y, g^{z_2 z_4}, g^{z_3+z_4}] \in \mathbb{G}^6$, where z_1, z_2, z_3, z_4 are picked at random from $\mathbb{Z}_p$, decide whether $Y = g^{z_1+z_3}$. It is easy to show that this is equivalent to the original D-Linear problem. For convenience, let $g_1 = g^{z_1}$, $g_2 = g^{z_2}$, $g_{24}^{\times} = g^{z_2 z_4}$, $g_{34}^{+} = g^{z_3+z_4}$.

Without loss of generality, we show only how to prove step (d_1, l_1, n_1) of the hybrid argument.

Lemma 3.7.8 *Suppose* $\mathbf{G}$ *satisfies the* (τ, ϵ) *D-Linear assumption, then no adversary making* q *decryption key queries, within time* $\tau - \Theta(qD \log T)$*, can distinguish between* $\mathbb{W}_{d_1,l_1,n_1}$ *and the preceding game with more than* $\epsilon + 1/p$ *probability.*

Proof of Lemma 3.7.8: Let $\varphi_1 = (d_1, l_1)$. We describe a reduction such that if $Y = g^{z_1+z_3}$, then the simulator produces a ciphertext in which the block $[c^{(b)}_{(d_1,l_1),n_1}, c^{(a)}_{(d_1,l_1),n_1}]$ is well-formed; otherwise, if Y is picked at random, the block is random as well. Hence, if the adversary can distinguish between the two scenarios, the simulator can solve the D-Linear problem.

Init: The adversary selects a point $\mathbf{X}^*$ in space that it wishes to attack. Define $\mathcal{I}^*_\varphi = \mathcal{I}_\varphi(\mathbf{X}^*)$.

Setup: To create public and private parameters, the simulator does the following:

1. Pick the following parameters at random from $\mathbb{Z}_p^{12DL-3}$:

$$\omega, \quad \left[\alpha_{\varphi,n}, \beta_{\varphi,n}, \bar{\theta}_{\varphi,n}, \bar{\theta}'_{\varphi,n}\right]_{\varphi=(d,l)\in[D]\times[L], n\in[2], (\varphi,n)\neq(\varphi_1,n_1)}, \quad \left[\theta_{\varphi,n}, \theta'_{\varphi,n}\right]_{\varphi=(d,l)\in[D]\times[L], n\in[2]}$$

   subject to the constraint that

$$\left[\bar{\theta}_{\varphi,n}\mathcal{I}^*_\varphi + \bar{\theta}'_{\varphi,n} = 0\right]_{\varphi=(d,l)\in[D]\times[L], n\in[2], (\varphi,n)\neq(\varphi_1,n_1)}$$

   where $\mathcal{I}^*_\varphi = \mathcal{I}_\varphi(\mathbf{X}^*)$.

   We require that the α's, βs, $\bar{\theta}$'s and $\bar{\theta}'$'s are forcibly non-zero. In addition, later in Equation (3.5), we will need that $\theta_{\varphi_1,n_1}\mathcal{I}^*_{\varphi_1} + \theta'_{\varphi_1,n_1} \neq 0$. Hence, the simulator simply aborts if it happens to pick θ such that $\theta_{\varphi_1,n_1}\mathcal{I}^*_{\varphi_1} + \theta'_{\varphi_1,n_1} = 0$. Note that this happens with probability $1/p$, and this explains why the $1/p$ additive factor exists in the adversary's advantage in Lemma 3.7.8.

2. Compute and release to the adversary the following public parameters:

$$\Omega \leftarrow \mathbf{e}(g,g)^\omega, a_{\varphi_1,n_1} \leftarrow {g_1}^{\theta_{\varphi_1,n_1}}, b_{\varphi_1,n_1} \leftarrow {g_2}^{\theta_{\varphi_1,n_1}}, a'_{\varphi_1,n_1} \leftarrow {g_1}^{\theta'_{\varphi_1,n_1}}, b'_{\varphi_1,n_1} \leftarrow {g_2}^{\theta'_{\varphi_1,n_1}},$$
$$\left[\begin{array}{l} a_{\varphi,n} \leftarrow \left(g^{\theta_{\varphi,n}} g_1^{\bar{\theta}_{\varphi,n}}\right)^{\alpha_{\varphi,n}}, b_{\varphi,n} \leftarrow \left(g^{\theta_{\varphi,n}} g_1^{\bar{\theta}_{\varphi,n}}\right)^{\beta_{\varphi,n}}, \\ a'_{\varphi,n} \leftarrow \left(g^{\theta'_{\varphi,n}} g_1^{\bar{\theta}'_{\varphi,n}}\right)^{\alpha_{\varphi,n}}, b'_{\varphi,n} \leftarrow \left(g^{\theta'_{\varphi,n}} g_1^{\bar{\theta}'_{\varphi,n}}\right)^{\beta_{\varphi,n}} \end{array}\right]_{\varphi=(d,l)\in[D]\times[L], n\in[2], (\varphi,n)\neq(\varphi_1,n_1)}$$

   This posits that $\alpha_{\varphi_1,n_1} = z_1$, $\beta_{\varphi_1,n_1} = z_2$, both of which are unknown to the simulator.

3. Compute what it can of the private key:

$$\widetilde{\omega} \leftarrow g^\omega, \mathsf{a}_{\varphi_1,n_1} \leftarrow g_1, \mathsf{b}_{\varphi_1,n_1} \leftarrow g_2,$$
$$\left[\begin{array}{ll} \mathsf{a}_{\varphi,n} \leftarrow g^{\alpha_{\varphi,n}}, & \mathsf{b}_{\varphi,n} \leftarrow g^{\beta_{\varphi,n}}, \\ y_{\varphi,n} \leftarrow \left(g^{\theta_{\varphi,n}} g_1^{\bar{\theta}_{\varphi,n}}\right)^{\alpha_{\varphi,n}\beta_{\varphi,n}}, & y'_{\varphi,n} \leftarrow \left(g^{\theta'_{\varphi,n}} g_1^{\bar{\theta}'_{\varphi,n}}\right)^{\alpha_{\varphi,n}\beta_{\varphi,n}} \end{array}\right]_{\varphi=(d,l)\in[D]\times[L], n\in[2], (\varphi,n)\neq(\varphi_1,n_1)}$$

   Note that the simulator does not know y_{φ_1,n_1} and y'_{φ_1,n_1}.

The following lemma shows that even if we do not know the parameters z_1, z_2, y_{φ_1,n_1} or y'_{φ_1,n_1}, we can still compute certain terms efficiently.

Lemma 3.7.9 *In step (d_1, l_1, n_1) of the hybrid argument, let $\varphi_1 = (d_1, l_1)$. Suppose we are given $(d_2, l_2, n_2) \neq (d_1, l_1, n_1)$, and let $\varphi_2 = (d_2, l_2)$. Suppose ID_1 and ID_2 are nodes such that $\Phi(ID_1) = \varphi_1$ and $\Phi(ID_2) = \varphi_2$ and $ID_2 \neq \mathcal{I}^*_{\varphi_2}$. Moreover, suppose we are given $\lambda_1 \in \mathbb{Z}_p$. Then, even though the simulator does know y_{φ_1,n_1}, it can efficiently generate the following term, such that the its resulting distribution is the same as when λ_2 is picked uniformly at random.*

$$\left(y^{ID_1}_{\varphi_1,n_1} y'_{\varphi_1,n_1}\right)^{\lambda_1} \cdot \left(y^{ID_2}_{\varphi_2,n_2} y'_{\varphi_2,n_2}\right)^{\lambda_2} \tag{3.3}$$

Moreover, the following two terms can also be computed efficiently

$$\mathsf{a}^{-\lambda_2}_{\varphi_2,n_2}, \mathsf{b}^{-\lambda_2}_{\varphi_2,n_2}. \tag{3.4}$$

Proof: For simplicity, let $\alpha = \alpha_{\varphi_2,n_2}$, $\beta = \beta_{\varphi_2,n_2}$. For $i \in [2]$, we use simply θ_i to denote θ_{φ_i,n_i}, and θ'_i to denote θ'_{φ_i,n_1}. We use simply $\bar{\theta}_2$ to denote $\bar{\theta}_{\varphi_2,n_2}$, and $\bar{\theta}'_2$ to denote $\bar{\theta}'_{\varphi_2,n_2}$. Notice we do not define $\bar{\theta}_1$, since $\bar{\theta}_{\varphi_1,n_1}$ and $\bar{\theta}'_{\varphi_1,n_1}$ are not defined. Define for $i \in [2]$, $\Theta_i = \theta_i \cdot ID_i + \theta'_i$ and define $\overline{\Theta}_2 = \bar{\theta}_2 \cdot ID_2 + \bar{\theta}'_2$.

Recall that the simulator picked parameters such that $\bar{\theta}_2 \mathcal{I}^*_{\varphi_2} + \bar{\theta}'_2 = 0$. In addition, since $ID_2 \neq \mathcal{I}^*_{\varphi_2}$, and $\bar{\theta}_2 \neq 0$,

$$\overline{\Theta}_2 = \bar{\theta}_2 \cdot ID_2 + \bar{\theta}'_2 \neq 0$$

First, the simulator pick λ uniformly at random and define

$$\lambda_2 = \lambda - \frac{z_2 \lambda_1 \Theta_1}{\alpha\beta\overline{\Theta}_2}.$$

Observe that λ_2 is distributed uniformly, but we cannot compute λ_2 efficiently because we do not know z_2. However, since we know g^{z_2}, we can compute g^{λ_2} efficiently. Hence, it follows that we can compute the two terms in (3.4) efficiently in the following way.

$$\mathsf{a}^{-\lambda_2}_{\varphi_2,n_2} = (g^{\lambda_2})^{-\alpha}, \mathsf{b}^{-\lambda_2}_{\varphi_2,n_2} = (g^{\lambda_2})^{-\beta}.$$

It remains to show how to compute the term in (3.3). Rewrite (3.3) as below:

$$\left(y^{ID_1}_{\varphi_1,n_1} y'_{\varphi_1,n_1}\right)^{\lambda_1} \cdot \left(y^{ID_2}_{\varphi_2,n_2} y'_{\varphi_2,n_2}\right)^{\lambda_2} = g^{z_1 z_2 \lambda_1 \left(\theta_1 ID_1 + \theta'_1\right)} \cdot \left(g^{\alpha\beta(\theta_2 + z_1\bar{\theta}_2) ID_2} g^{\alpha\beta(\theta'_2 + z_1\bar{\theta}'_2)}\right)^{\lambda_2}$$
$$= g^{z_1 z_2 \lambda_1 \Theta_1 + \alpha\beta(\Theta_2 + z_1\overline{\Theta}_2)(\lambda - z_2\lambda_1\Theta_1/\alpha\beta\overline{\Theta}_2)} = g^{\alpha\beta\Theta_2\lambda} \cdot (g^{z_1})^{\alpha\beta\overline{\Theta}_2\lambda} \cdot (g^{z_2})^{-\lambda_1\Theta_1\Theta_2/\overline{\Theta}_2},$$

which can be computed efficiently given g^{z_1} and g^{z_2}. ■

Phase 1: Suppose the adversary makes a decryption query for the hyper-rectangle $\mathbf{B}(s_1, t_1, \ldots, s_D, t_D)$. Since $\mathbf{B}$ does not contain $\mathbf{X}^*$, there exists a dimension $d_0 \in [D]$ such that $x^*_{d_0} \notin [s_{d_0}, t_{d_0}]$, where $x^*_{d_0}$ is $\mathbf{X}^*$ projected onto the d_0^{th} dimension. Hence, exactly one of the following cases must be true:

Case 1: For all $ID \in \Lambda_{d_1}(\mathbf{B})$ such that $\Phi(ID) = \varphi_1$, $ID \neq \mathcal{I}_{\varphi_1}(\mathbf{X}^*)$.

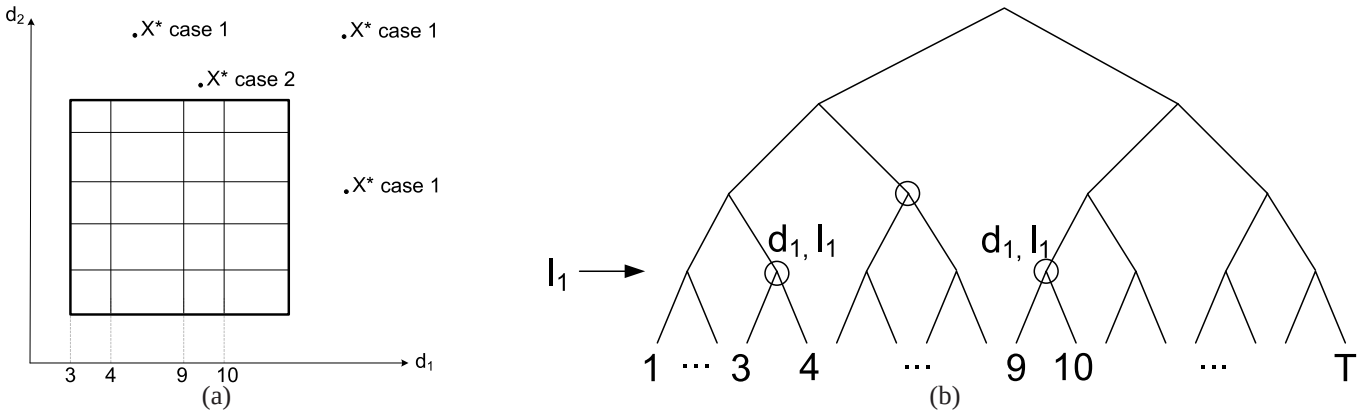


Figure 3.3: A 2-dimensional example: Relative position between $\mathbf{X}^*$ and the queried hyper-rectangle. ***(a)*** *Each small rectangle shown is a simple rectangle. Along dimension d_1, ranges $[3,4]$ and $[9,10]$ correspond to nodes at level l_1.* ***(b)*** *The interval tree corresponding to dimension d_1.*

Case 2: There exists $ID \in \Lambda_{d_1}(\mathbf{B})$ such that $\Phi(ID) = \varphi_1$ and $ID = \mathcal{I}_{\varphi_1}(\mathbf{X}^*)$. Note that in this case, for all $ID' \in \Lambda_{d_1}(\mathbf{B})$ such that $ID' \neq ID$, $ID' \neq \mathcal{I}_{\varphi'}(\mathbf{X}^*)$, where $\varphi' = \Phi(ID')$; moreover, there exists a dimension d_0, such that for all $ID_0 \in \Lambda_{d_0}(\mathbf{B})$, $ID_0 \neq \mathcal{I}_{\varphi_0}(\mathbf{X}^*)$, where $\varphi_0 = \Phi(ID_0)$.

Figure 3.3 illustrates the above two cases with a 2-dimensional example. We now explain how the simulator generates the decryption key in each of the above cases.

Case 1: (a) Pick at random $[\widetilde{\mu}_d]_{d\in[D]} \in_R \mathbb{G}^D$, such that $\prod_{d\in[D]} \widetilde{\mu}_d = \widetilde{\omega}$.

(b) For each $ID \in \Lambda^{\cup}(\mathbf{B})$ where $\varphi := \Phi(ID) \neq \varphi_1$, pick at random $\lambda_{ID,1}, \lambda_{ID,2}$. Let $\mathbf{DK}(ID) = \left(\mathsf{k}_{ID,0}, [\mathsf{k}^{(\mathsf{a})}_{ID,1}, \mathsf{k}^{(\mathsf{b})}_{ID,1}], [\mathsf{k}^{(\mathsf{a})}_{ID,2}, \mathsf{k}^{(\mathsf{b})}_{ID,2}]\right)$ represent the element in $\mathbf{DK}$ for ID, compute and release $\mathbf{DK}(ID)$ as below:

$$\mathsf{k}_{ID,0} \leftarrow \widetilde{\mu}_d \cdot \prod_{n\in[2]} \left(y^{ID}_{\varphi,n} y'_{\varphi,n}\right)^{\lambda_{ID,n}},$$
$$\left[\mathsf{k}^{(\mathsf{a})}_{ID,n} \leftarrow \mathsf{a}^{-\lambda_{ID,n}}_{\varphi,n}, \mathsf{k}^{(\mathsf{b})}_{ID,n} \leftarrow \mathsf{b}^{-\lambda_{ID,n}}_{\varphi,n}\right]_{n\in[2]}$$

(c) For each $ID \in \Lambda^{\cup}(\mathbf{B})$ such that $\Phi(ID) = \varphi_1$, the simulator can compute the following $\mathbf{DK}(ID)$ efficiently:

$$\mathsf{k}_{ID,0} \leftarrow \widetilde{\mu}_{d_1} \cdot \prod_{n\in[2]} \left(y^{ID}_{\varphi_1,n} y'_{\varphi_1,n}\right)^{\lambda_{ID,n}},$$
$$\left[\mathsf{k}^{(\mathsf{a})}_{ID,n} \leftarrow \mathsf{a}^{-\lambda_{ID,n}}_{\varphi_1,n}, \mathsf{k}^{(\mathsf{b})}_{ID,n} \leftarrow \mathsf{b}^{-\lambda_{ID,n}}_{\varphi_1,n}\right]_{n\in[2]}$$

Since the simulator does not know y_{φ_1,n_1} or y'_{φ_1,n_1}, it needs to use Lemma 3.7.9 to generate $\mathbf{DK}(ID)$. Let $n' \neq n_1$. To apply Lemma 3.7.9, the simulator first picks at

random λ_{ID,n_1}, and rewrites $\mathsf{k}_{ID,0}$ as

$$\mathsf{k}_{ID,0} = \widetilde{\mu}_{d_1} \cdot \left(y^{ID}_{\varphi_1,n_1} y'_{\varphi_1,n_1}\right)^{\lambda_{ID,n_1}} \cdot \left(y^{ID}_{\varphi_1,n'} y'_{\varphi_1,n'}\right)^{\lambda_{ID,n'}}$$

Since $ID \neq \mathcal{I}_{\varphi_1}(\mathbf{X}^*)$, the simulator can apply Lemma 3.7.9 by substituting (d_2, l_2, n_2) in the lemma with (d_1, l_1, n'), and λ_1 with λ_{ID,n_1}; in addition, both ID_1 and ID_2 in the lemma are substituted with ID.

Case 2: (a) Pick at random $[\mu_d]_{d\in[D]} \in_R \mathbb{Z}_p$ such that $\sum_{d\in[D]} \mu_d = \omega$.

(b) For each $ID \in \Lambda^{\cup}(\mathbf{B}) - \Lambda_{d_0}(\mathbf{B}) - \Lambda_{d_1}(\mathbf{B})$ where $\varphi := \Phi(ID) = (d, l)$, $d \neq d_0$ and $d \neq d_1$, pick at random $\lambda_{ID,1}, \lambda_{ID,2}$. Let $\mathbf{DK}(ID) = \left(\mathsf{k}_{ID,0}, [\mathsf{k}^{(\mathsf{a})}_{ID,1}, \mathsf{k}^{(\mathsf{b})}_{ID,1}], [\mathsf{k}^{(\mathsf{a})}_{ID,2}, \mathsf{k}^{(\mathsf{b})}_{ID,2}]\right)$ represent the element in $\mathbf{DK}$ for ID, compute and release $\mathbf{DK}(ID)$ as below:

$$\mathsf{k}_{ID,0} \leftarrow g^{\mu_d} \cdot \prod_{n\in[2]} \left(y^{ID}_{\varphi,n} y'_{\varphi,n}\right)^{\lambda_{ID,n}}$$
$$\left[\mathsf{k}^{(\mathsf{a})}_{ID,n} \leftarrow \mathsf{a}^{-\lambda_{ID,n}}_{\varphi,n}, \mathsf{k}^{(\mathsf{b})}_{ID,n} \leftarrow \mathsf{b}^{-\lambda_{ID,n}}_{\varphi,n}\right]_{n\in[2]}$$

(c) Let $\overline{ID} \in \Lambda_{d_1}(\mathbf{B})$ and $\overline{ID} = \mathcal{I}_{\varphi_1}(\mathbf{X}^*)$. There exists exactly one such $\overline{ID}$. The simulator picks at random $\lambda_{\overline{ID},n_1} \in_R \mathbb{Z}_p$. Define $\Upsilon = \left(y^{\overline{ID}}_{\varphi_1,n_1} y'_{\varphi_1,n_1}\right)^{\lambda_{\overline{ID},n_1}}$.

(d) For each $ID \in \Lambda_{d_0}(\mathbf{B})$ where $\varphi_0 = (d_0, l) := \Phi(ID)$, compute and release $\mathbf{DK}(ID)$:

$$\mathsf{k}_{ID,0} \leftarrow g^{\mu_{d_0}} \cdot \Upsilon \cdot \prod_{n\in[2]} \left(y^{ID}_{\varphi_0,n} y'_{\varphi_0,n}\right)^{\lambda_{ID,n}}$$
$$\left[\mathsf{k}^{(\mathsf{a})}_{ID,n} \leftarrow \mathsf{a}^{-\lambda_{ID,n}}_{\varphi_0,n}, \mathsf{k}^{(\mathsf{b})}_{ID,n} \leftarrow \mathsf{b}^{-\lambda_{ID,n}}_{\varphi_0,n}\right]_{n\in[2]}$$

This implies that $\widetilde{\mu}_{d_0} = g^{\mu_{d_0}} \cdot \Upsilon$. Note that Υ cannot be computed efficiently, as the simulator does not know y_{φ_1,n_1} or y'_{φ_1,n_1}. However, since $ID \neq \mathcal{I}_{\varphi_0}(\mathbf{X}^*)$, the simulator can apply Lemma 3.7.9 by substituting (d_2, l_2, n_2) in the lemma with $(d_0, l, 1)$, λ_1 with $\lambda_{\overline{ID},n_1}$, ID_1 with $\overline{ID}$, and ID_2 with ID. The remaining terms in $\mathsf{k}_{ID,0}$ can be computed efficiently.

(e) For each $ID \in \Lambda_{d_1}(\mathbf{B})$ where $\varphi'_1 = (d_1, l) := \Phi(ID) \neq \varphi_1$, compute and release $\mathbf{DK}(ID)$:

$$\mathsf{k}_{ID,0} \leftarrow g^{\mu_{d_1}} \cdot \Upsilon^{-1} \cdot \prod_{n\in[2]} \left(y^{ID}_{\varphi'_1,n} y'_{\varphi'_1,n}\right)^{\lambda_{ID,n}}$$
$$\left[\mathsf{k}^{(\mathsf{a})}_{ID,n} \leftarrow \mathsf{a}^{-\lambda_{ID,n}}_{\varphi'_1,n}, \mathsf{k}^{(\mathsf{b})}_{ID,n} \leftarrow \mathsf{b}^{-\lambda_{ID,n}}_{\varphi'_1,n}\right]_{n\in[2]}$$

This implies that $\widetilde{\mu}_{d_1} = g^{\mu_{d_1}} \cdot \Upsilon^{-1}$. Note that Υ^{-1} cannot be computed efficiently, as the simulator does not know y_{φ_1,n_1} or y'_{φ_1,n_1}. However, since $ID \neq \mathcal{I}_{\varphi'_1}(\mathbf{X}^*)$, the simulator can apply Lemma 3.7.9, by substituting (d_2, l_2, n_2) in the lemma with $(d_1, l, 1)$, λ_1 with $-\lambda_{\overline{ID},n_1}$, ID_1 with $\overline{ID}$, and ID_2 with ID. The remaining terms in $\mathsf{k}_{ID,0}$ can be computed efficiently.

(f) For $\overline{ID}$, let $n' \neq n_1$. Pick $\lambda_{\overline{ID},n'}$ at random from $\mathbb{Z}_p$. Then compute and release the following $\mathbf{DK}(\overline{ID})$:

$$\mathsf{k}_{\overline{ID},0} \leftarrow g^{\mu_{d_1}} \cdot \Upsilon^{-1} \cdot \prod_{n\in[2]} \left(y^{\overline{ID}}_{\varphi_1,n} y'_{\varphi_1,n}\right)^{\lambda_{\overline{ID},n}},$$
$$\left[\mathsf{k}^{(\mathsf{a})}_{\overline{ID},n} \leftarrow \mathsf{a}^{-\lambda_{\overline{ID},n}}_{\varphi_1,n}, \mathsf{k}^{(\mathsf{b})}_{\overline{ID},n} \leftarrow \mathsf{b}^{-\lambda_{\overline{ID},n}}_{\varphi_1,n}\right]_{n\in[2]}$$

As before, here $\widetilde{\mu}_{d_1} = g^{\mu_{d_1}} \cdot \Upsilon^{-1}$. $\mathsf{k}_{\overline{ID},0}$ can be computed because the terms containing y_{φ_1,n_1} and y'_{φ_1,n_1} cancel out, leaving $\mathsf{k}_{\overline{ID},0} = g^{\mu_{d_1}} \cdot \left(y^{\overline{ID}}_{\varphi_1,n'} y'_{\varphi_1,n'}\right)^{\lambda_{\overline{ID},n'}}$.

(g) For each $ID \in \Lambda_{d_1}(\mathbf{B})$ such that $\Phi(ID) = \varphi_1$ and $ID \neq \overline{ID}$, compute and release $\mathbf{DK}(ID)$:

$$\mathsf{k}_{ID,0} \leftarrow g^{\mu_{d_1}} \cdot \Upsilon^{-1} \cdot \prod_{n\in[2]} \left(y^{ID}_{\varphi_1,n} y'_{\varphi_1,n}\right)^{\lambda_{ID,n}},$$
$$\left[\mathsf{k}^{(\mathsf{a})}_{ID,n} \leftarrow \mathsf{a}^{-\lambda_{ID,n}}_{\varphi_1,n}, \mathsf{k}^{(\mathsf{b})}_{ID,n} \leftarrow \mathsf{b}^{-\lambda_{ID,n}}_{\varphi_1,n}\right]_{n\in[2]}$$

Again, to be able to generate $\mathsf{k}_{ID,0}$, Lemma 3.7.9 is required. However, in this case, a slight complication is involved, since two terms in $\mathsf{k}_{ID,0}$ contain y_{φ_1,n_1} and y'_{φ_1,n_1}:

$$\begin{aligned}
\mathsf{k}^{(\mathrm{O})}_{ID} &= g^{\mu_{d_1}} \cdot \Upsilon^{-1} \cdot \prod_{n\in[2]} \left(y^{ID}_{\varphi_1,n} y'_{\varphi_1,n}\right)^{\lambda_{ID,n}} \\
&= g^{\mu_{d_1}} \cdot \left(y^{\overline{ID}}_{\varphi_1,n_1} y'_{\varphi_1,n_1}\right)^{-\lambda_{\overline{ID},n_1}} \cdot \prod_{n\in[2]} \left(y^{ID}_{\varphi_1,n} y'_{\varphi_1,n}\right)^{\lambda_{ID,n}} \\
&= g^{\mu_{d_1}} \cdot \left(\left(y^{\overline{ID}}_{\varphi_1,n_1} y'_{\varphi_1,n_1}\right)^{-\lambda_{\overline{ID},n_1}} \cdot \left(y^{ID}_{\varphi_1,n_1} y'_{\varphi_1,n_1}\right)^{\lambda_{ID,n_1}}\right) \cdot \left(y^{ID}_{\varphi_1,n'} y'_{\varphi_1,n'}\right)^{\lambda_{ID,n'}}
\end{aligned}$$

Now the simulator picks λ_{ID,n_1} at random from $\mathbb{Z}^*_p$, and computes

$$\widetilde{\lambda}_{ID,n_1} = \lambda_{ID,n_1} \frac{\theta_{\varphi_1,n_1} \cdot ID + \theta'_{\varphi_1,n_1}}{\theta_{\varphi_1,n_1} \cdot \overline{ID} + \theta'_{\varphi_1,n_1}} - \lambda^{(\overline{ID})}_{n_1} \tag{3.5}$$

Here we require that $\theta_{\varphi_1,n_1} \cdot \overline{ID} + \theta'_{\varphi_1,n_1} \neq 0$. Notice that $\overline{ID} = \mathcal{I}^*_{\varphi_1}$. As we explained in the **Setup** stage, the simulator aborts if it happens to pick $\theta_{\varphi_1,(n_1,j)}$'s such that $\theta_{\varphi_1,n_1}\mathcal{I}^*_{\varphi_1} + \theta'_{\varphi_1,n_1} = 0$. Hence,

$$\mathsf{k}_{ID,0} = g^{\mu_{d_1}} \cdot \left(y^{\overline{ID}}_{\varphi_1,n_1} y'_{\varphi_1,n_1}\right)^{\widetilde{\lambda}_{ID,n_1}} \cdot \left(y^{ID}_{\varphi_1,n'} y'_{\varphi_1,n'}\right)^{\lambda_{ID,n'}}$$

And now the simulator can apply Lemma 3.7.9 by substituting (d_2, l_2, n_2) in the lemma with (d_1, l_1, n'), λ_1 with $\widetilde{\lambda}_{ID,n_1}$, ID_1 with $\overline{ID}$, and ID_2 with ID.

Challenge: On receiving a message Msg from the adversary, the simulator does the following:

1. Pick random integers $[r_{\varphi,n}]_{\varphi=(d,l)\in[D]\times[L],n\in[2]} \in \mathbb{Z}_p^{2DL}$.
2. Compute and release the following as the ciphertext.

$$*, g_{34}^{+}, [*,*], \ldots, [*,*], \quad \left(g_{24}^{\times}\right)^{\theta_{\varphi_1,n_1}\mathcal{I}^*_{\varphi_1}+\theta'_{\varphi_1,n_1}}, \quad Y^{\theta_{\varphi_1,n_1}\mathcal{I}^*_{\varphi_1}+\theta'_{\varphi_1,n_1}},$$
$$\left[g^{r_{\varphi,n}\beta_{\varphi,n}\left(\theta_{\varphi,n}\mathcal{I}^*_{\varphi}+\theta'_{\varphi,n}\right)}, \quad \left(g_{34}^{+}\cdot g^{-r_{\varphi,n}}\right)^{\alpha_{\varphi,n}\left(\theta_{\varphi,n}\mathcal{I}^*_{\varphi}+\theta'_{\varphi,n}\right)}\right]_{(d_1,l_1,n_1)<(d,l,n)<(D,L,2),\varphi=(d,l)}$$

   where $(d,l,n) < (d',l',n')$ if and only if 1) $d < d'$; or 2) $d = d'$ and $l < l'$; or 3) $(d,l) = (d',l')$ and $n < n'$.

Note that this implies that $r = z_3 + z_4$ and $r_{\varphi_1,n_1} = z_4$. If $Y = g^{z_1 z_3}$, it is easy to verify that the ciphertext is well-formed, due to the fact that

$$\left[\bar{\theta}_{\varphi,n}\mathcal{I}^*_{\varphi} + \bar{\theta}'_{\varphi,n} = 0\right]_{(d,l,n)\neq(d_1,l_1,n_1),\varphi=(d,l)}$$

If Y is a random number, then term $c^{(a)}_{(d_1,l_1),n_1}$ is random and independent of the remaining terms of the ciphertext.

Phase 2: Phase 1 is repeated.

Guess: If the adversary guesses that the ciphertext is an encryption of $\mathbf{Msg}$ under $\mathbf{X}^*$, the simulator guesses that $Y = g^{z_3+z_4}$. Else if the adversary guesses that the ciphertext is the encryption under a random point, then the simulator guesses that Y is picked at random from $\mathbb{G}$. ■

Proof of Theorem 3.7.7: The theorem follows naturally from Lemma 3.7.8 and the hybrid argument. ■

Chapter 4

Delegating Capabilities in Predicate Encryption

In this chapter, we demonstrate how to add delegation to predicate encryption systems. We first give the formal definition for delegation in predicate encryption, including definitions of security. While our big goal is to support expressive query predicates, as an initial step towards this vision, we shall first add delegation to predicate systems supporting conjunctive queries. In particular, we add delegation to an HVE-like construction; and we call the new scheme Delegatable Hidden Vector Encryption (dHVE). The technical contents of this paper has been published in ICALP 2008 [37].

4.1 Definitions

We introduce the notion of delegation in predicate encryption systems and provide a formal definition of security.

In a predicate encryption system, some user, Alice, creates a public key and a corresponding master key. Using her master key, Alice can compute and hand out a token to Bob, such that Bob is able to evaluate some function[1], f, on the plaintext that has been encrypted. Meanwhile, Bob cannot learn any more information about the plaintext, apart from the output of the function f.

In this thesis, we consider the role of delegation in predicate encryption systems. Suppose Alice (the master key owner) has given Bob tokens to evaluate a set of functions $f_1, f_2, \ldots, f_m$ over ciphertexts. Now Bob wishes to delegate to Charles the ability to evaluate the functions $\{f_1 + f_2, f_3, f_4\}$ over the ciphertext. Charles should not be able to learn more information about the plaintext apart from the output of the functions $\{f_1 + f_2, f_3, f_4\}$. For example, although Charles can evaluate $f_1 + f_2$, he should not be able to learn f_1 or f_2 separately. In general, Bob may be interested in delegating any set of functions that is more *restrictive* than what he is able to evaluate with his tokens. In general, a user who has a delegated capability can in turn create an even more

[1] Although we focus on functions that are predicates in our solutions, we use the more general term of functions in this discussion and our formal definitions.

restricted capability. For example, after obtaining a token from Bob for functions $\{f_1 + f_2, f_3, f_4\}$, Charles may now decide to delegate to his friend David a token to evaluate $f_3 \cdot f_4$.

4.1.1 Definition

We now formally define delegation in predicate encryption systems that captures the above notion.

Let $X = (x_1, x_2, \ldots, x_\ell) \in \{0,1\}^\ell$ denote a plaintext. Without loss of generality, assume that we would like to evaluate from the ciphertext boolean functions (a.k.a. predicates) on X. Functions that output multiple bits can be regarded as concatenation of boolean functions. Let $\mathcal{F}$ denote the set of all boolean functions from $\{0,1\}^\ell$ to $\{0,1\}$, i.e., $\mathcal{F} := \{f \mid f : \{0,1\}^\ell \to \{0,1\}\}$.

We define a token as a capability that allows one to evaluate from the ciphertext a set of functions on X. Tokens will be associated with a set $\mathcal{G} = \{g_1, g_2, \ldots, g_m\} \subseteq \mathcal{F}$ that can compute a subset of all available functions. We remark that a token might be represented much more succintly than $|\mathcal{G}|$. For instance, if one had the capability to learn each individual bit of X one could have a small token, but still compute all 2^{2^ℓ} predicate functions on the input.

A delegatable Predicate Encryption (DPE) scheme consists of the following (possibly randomized) algorithms.

$Setup(1^\lambda)$ The $Setup$ algorithm takes as input a security parameter 1^λ and outputs a public key PK and a master secret key MSK.

$Encrypt(\mathsf{PK}, X)$ The $Encrypt$ algorithm takes as input a public key PK and a plaintext $X = (x_1, x_2, \ldots, x_\ell) \in \{0,1\}^\ell$ and outputs a ciphertext CT.

$GenToken(\mathsf{PK}, \mathsf{MSK}, \mathcal{G})$ The $GenToken$ algorithm takes as input a public key PK, master secret key MSK, and a set of boolean functions $\mathcal{G} \subseteq \mathcal{F}$. It outputs a token for evaluating the set of functions $\mathcal{G}$ from a ciphertext.

$Query(\mathsf{PK}, \mathsf{TK}_\mathcal{G}, \mathsf{CT}, f)$ The $Query$ algorithm takes as input a public key PK, a token $\mathsf{TK}_\mathcal{G}$ for the function family $\mathcal{G}$, a function $f \in \mathcal{G}$, and a ciphertext CT. If CT is an encryption of the plaintext X, then the algorithm outputs $f(X)$.

$Delegate(\mathsf{PK}, \mathsf{TK}_\mathcal{G}, \mathcal{G}')$ The $Delegate$ algorithm takes as input a public key PK, a token for the function family $\mathcal{G} \subseteq \mathcal{F}$, and $\mathcal{G}' \subseteq \mathcal{G}$. It computes a token for evaluating the function family $\mathcal{G}'$ on a ciphertext.

Remark 4.1.1 *We note that the above definition captures delegation in predicate encryption systems in the broadest sense. In a predicate encryption system, we would like to maximize the expressiveness of delegation; however, one should not be able to delegate beyond what she can learn with her own tokens. Otherwise, the security of predicate encryption would be broken.*

Since we care about being able to perform expressive delegations, we can judge a system by its expressiveness, e.g., what types of functions one can evaluate over the ciphertext, and what types of delegations one can perform. Our vision is to design a predicate encryption system that supports a rich set of queries and delegations. As an initial step, we restrict ourselves to some special classes of functions. At the time this research is being conducted, the most expressive predicate encryption system (without delegation) we know of supports conjunctive queries [12]; we focus our efforts on permitting delegation in such systems.

More recently, Katz, Sahai, and Waters proposed a novel predicate encryption system supporting inner product queries [28] and realized a more expressive system. An interesting open direction is to figure out what types of delegation one might realize in their system.

4.1.2 Security

We now define the security for delegation in predicate encryption systems. We describe a query security game between a challenger and an adversary. This game formally captures the notion that the tokens reveal no unintended information about the plaintext. The adversary asks the challenger for a number of tokens. For each queried token, the adversary gets to specify its path of derivation: whether the token is directly generated by the root authority, or delegated from another token. If the token is delegated, the adversary also gets to specify from which token it is delegated. The game proceeds as follows:

Setup. The challenger runs the $Setup$ algorithm and gives the adversary the public key PK.

Query 1. The adversary adaptively makes a polynomial number of queries of the following types:

- *Create token.* The adversary asks the challenger to create a token for a set of functions $\mathcal{G} \subseteq \mathcal{F}$. The challenger creates a token for $\mathcal{G}$ without giving it to the adversary.
- *Create delegated token.* The adversary specifies a token for function family $\mathcal{G}$ that has already been created, and asks the challenger to perform a delegation operation to create a child token for $\mathcal{G}' \subseteq \mathcal{G}$. The challenger computes the child token without releasing it to the adversary.
- *Reveal token.* The adversary asks the challenger to reveal an already-created token for function family $\mathcal{G}$.

Note that when token creation requests are made, the adversary does not automatically see the created token. The adversary sees a token only when it makes a reveal token query.

Challenge. The adversary outputs two strings $X_0^*, X_1^* \in \{0,1\}^\ell$ subject to the following constraint:

For any token revealed to the adversary in the **Query 1** stage, let $\mathcal{G}$ denote the function family corresponding to this token. For all $f \in \mathcal{G}$, $f(X_0^*) = f(X_1^*)$.

Next, the challenger flips a random coin b and encrypts X_b^*. It returns the ciphertext to the adversary.

Query 2. Repeat the **Query 1** stage. All tokens revealed in this stage must satisfy the same condition as above.

Guess. The adversary outputs a guess b' of b. The advantage of an adversary $\mathcal{A}$ in the above game is defined to be $\mathsf{Adv}_{\mathcal{A}} = |\Pr[b = b'] - 1/2|$.

Definition 4.1.1 *We say that a delegatable predicate encryption system is* secure *if for all polynomial-time adversaries $\mathcal{A}$ attacking the system, its advantage $\mathsf{Adv}_{\mathcal{A}}$ is a negligible function of λ.*

Selective security

We also define a weaker security notion called *selective security*. In the selective security game, instead of submitting two strings X_0^*, X_1^* in the **Challenge** stage, the adversary first commits to two strings at the beginning of the security game. The rest of the security game proceeds exactly as before. The selective security model has been used earlier in the literature [3, 12, 13, 14, 15, 35].

We say that a delegatable predicate encryption system is *selectively secure* if all polynomial time adversaries $\mathcal{A}$ have negligible advantage in the selective security game.

Remark 4.1.2 *We note that our security definition is complete in the sense that in the query phase, the adversary gets to specify, for each queried token, its path of derivation: whether the token is generated by the root authority, or from whom the token has been delegated. In prior work on delegation in identity-based encryption systems (e.g., Hierarchical Identity-Based Encryption (HIBE) [4], Anonymous Hierarchical Identity-Based Encryption (AHIBE) [13]), the security game was under-specified. In these definitions, the adversary did not get to specify from whom each queried token is delegated.*

One way to deal with this is to create systems where all tokens are generated from the same probability distribution. For instance, the AHIBE [13] work uses this approach. While this allows us to prove the security of these systems, it can be an overkill. Under our security definition, the delegated token need not be picked from the same probability distribution as the non-delegated tokens. In fact, we show that the ability to capture such nuances in our security definition allows us to construct a simpler AHIBE scheme with smaller private key size.

4.1.3 A simple example

To help understand the above definition, we give a simple example similar to that in the BW06 paper [12]. As shown by Figure 4.1, the point X encrypted takes on integer values between 0 and T. Given $a, b \in [0, T]$, let $f_{a,b}$ denote the function that decides whether $X \in [a, b]$:

$$f_{a,b}(X) = \begin{cases} 1 & X \in [a, b] \\ 0 & \text{o.w.} \end{cases}$$

In Figure 4.1, we mark three disjoint segments $[a_1, a_2]$, $[a_3, a_4]$, $[a_5, a_6]$ and four points x, y, z, u. Alice has a token for functions $\{f_{a_1,a_2}, f_{a_3,a_4}, f_{a_5,a_6}\}$. This allows her to evaluate the following three predicates: whether $a_1 \leq X \leq a_2$, $a_3 \leq X \leq a_4$, and $a_5 \leq X \leq a_6$. Alice can now distinguish between ciphertexts $Encrypt(\mathsf{PK}, x)$ and $Encrypt(\mathsf{PK}, y)$, but she cannot distinguish between ciphertexts $Encrypt(\mathsf{PK}, y)$ and $Encrypt(\mathsf{PK}, z)$.

Alice performs a delegation and computes a child token for the function $g(X) = f_{a_1,a_2}(X) \vee f_{a_3,a_4}(X)$, and Bob receives this delegated token from Alice. Bob can decide whether $(a_1 \leq X \leq a_2) \vee (a_3 \leq X \leq a_4)$; this is a subset of information allowed by Alice's token. Given this new token, Bob can decide whether X falls inside these two ranges, but he cannot decide between the cases whether $X \in [a_1, a_2]$ or $X \in [a_3, a_4]$. For example, Bob can distinguish between the ciphertexts $Encrypt(\mathsf{PK}, x)$ and $Encrypt(\mathsf{PK}, u)$, but he cannot distinguish between the ciphertexts $Encrypt(\mathsf{PK}, x)$ and $Encrypt(\mathsf{PK}, y)$.

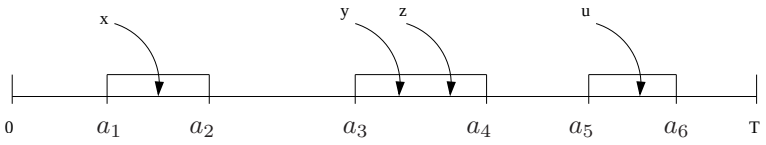


Figure 4.1: A simple example of predicate encryption similar to the one described in BW06 [12].

4.2 Delegatable Hidden Vector Encryption (dHVE)

We propose a primitive called delegatable hidden vector encryption (dHVE), where we add delegation to the HVE construction proposed in BW06 [12]. This is an interesting special case to the general definition given in Section 4.1.1, and represents an initial step toward our bigger vision of enabling expressive queries and delegations in predicate encryption systems.

4.2.1 Delegatable HVE overview (dHVE)

In our dHVE system, a plaintext consists of multiple "fields". For example, a plaintext can be the tuple (IP, PORT, TIME, LENGTH). A token corresponds to a conjunction of a subset of these fields: we can fix a field to a specific value, make a field "delegatable", or choose not to include a field in a query. For example, the query $(\text{IP} = ?) \wedge (\text{PORT} = 80) \wedge (\text{TIME} = 02/10/08)$ fixes the values of the PORT and TIME fields, and makes the IP field delegatable. The LENGTH field is not included in the query. A party in possession of this token can fill in any appropriate value for the delegatable field IP; however, she cannot change the values of a fixed field such as PORT or delete them from the query, nor can she add in the missing field LENGTH to the query. We now give formal definitions for the above notions.

Let Σ denote a finite alphabet and let $?, \perp$ denote two special symbols not in Σ. Define $\Sigma_{?,\perp} := \Sigma \cup \{?, \perp\}$. The symbol $?$ denotes a delegatable field, i.e., a field where one is allowed to fill in an arbitrary value and perform delegation. The symbol $\perp$ denotes a "don't care" field, i.e., a field not involved in some query. Typically, if a query predicate does not concern a specific field, we call this field a "don't care" field. In the aforementioned example, $(\text{IP} = ?) \wedge (\text{PORT} = 80) \wedge (\text{TIME} = 02/10/08)$, the IP field is delegatable, LENGTH is "don't care", and the remaining fields are fixed.

Plaintext Space. In dHVE, our plaintext is composed of a message $\mathbf{Msg} \in \{0,1\}^*$ and ℓ fields, denoted by $X = (x_1, x_2, \ldots, x_\ell) \in \Sigma^\ell$. Capabilities will be evaluated over X, and the $\mathbf{Msg}$ component is an extra message that will be divulged in case the predicate evaluates to true.

The *Encrypt* algorithm takes as input a public key PK, a pair $(X, \mathbf{Msg}) \in \{0,1\}^* \times \Sigma^\ell$, and outputs a ciphertext CT.

Tokens. In dHVE, a token allows one to evaluate a special class of boolean functions on the fields $X \in \Sigma^\ell$. We use a vector $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_\ell) \in (\Sigma_{?,\perp})^\ell$ to specify a set of functions being queried. Given σ, let $\mathcal{W}(\sigma)$ denote the indices of all delegatable fields, let $\mathcal{D}(\sigma)$ denote the indices of all "don't care" fields, and let $\mathcal{S}(\sigma)$ denote the indices of the remaining fixed fields. In the

following, we use the notation $[\ell]$ to denote the set $\{1, 2, \ldots, \ell\}$.

$$\mathcal{W}(\sigma) := \{i \mid \sigma_i = ?\}, \qquad \mathcal{D}(\sigma) := \{i \mid \sigma_i = \perp\}$$
$$\mathcal{S}(\sigma) := \{i \mid \sigma_i \in \Sigma\} = [\ell] \setminus (\mathcal{W}(\sigma) \cup \mathcal{D}(\sigma))$$

Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_\ell) \in (\Sigma_{?,\perp})^\ell$; σ specifies the following function family $\mathcal{C}_\sigma$ on the point $X = (x_1, \ldots, x_\ell)$ encrypted:

$$\mathcal{C}_\sigma := \left\{ \left(\bigwedge_{i \in W'} (x_i = a_i) \right) \wedge \left(\bigwedge_{j \in \mathcal{S}(\sigma)} (x_j = \sigma_j) \right) \mid W' \subseteq \mathcal{W}(\sigma), \forall i \in W', a_i \in \Sigma \right\} \tag{4.1}$$

In other words, given a token for σ, the family $\mathcal{C}_\sigma$ denotes the set of functions we can evaluate from a ciphertext. For the delegatable fields, we can fill in any appropriate value, but we cannot change or delete any of the fixed fields or add a "don't care" field to the query. If any function in $\mathcal{C}_\sigma$ evaluates to 1, one would also be able to decrypt the payload message $\mathbf{Msg}$.

Remark 4.2.1 *The family $\mathcal{C}_\sigma$ is a set of conjunctive equality tests, where we can fill in every delegatable field in σ with a value in Σ or "don't care". In particular, we fill in fields in W' with appropriate values in σ, and for the remaining delegatable fields $\mathcal{W}(\sigma) - W'$, we fill them with "don't care". If σ has no delegatable field, then the set $\mathcal{C}_\sigma$ contains a single function. This is exactly the case considered by the original HVE construction, where each token allows one to evaluate a single function from a ciphertext.*

Delegation. In dHVE, Alice, who has a token for σ, can delegate to Bob a subset of the functions she can evaluate: 1) Alice can fill in delegatable fields (i.e., $\mathcal{W}(\sigma)$) with a value in Σ or with the "don't care" symbol $\perp$; 2) Alice can also leave a delegatable field unchanged (with the ? symbol). In this case, Bob will be able to perform further delegation on that field.

Definition 4.2.1 *Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_\ell), \sigma' = (\sigma'_1, \sigma'_2, \ldots, \sigma'_\ell) \in \Sigma^\ell_{?,\perp}$. We say that $\sigma' \prec \sigma$, if for all $i \in \mathcal{S}(\sigma) \cup \mathcal{D}(\sigma)$, $\sigma'_i = \sigma_i$.*

Note that $\sigma' \prec \sigma$ means that from TK_σ we can perform a delegation operation and compute $\mathsf{TK}_{\sigma'}$. In addition, if $\sigma' \prec \sigma$, then $\mathcal{C}_{\sigma'} \subseteq \mathcal{C}_\sigma$, i.e., $\mathsf{TK}_{\sigma'}$ allows one to evaluate a subset of the functions allowed by TK_σ.

In summary, we introduce delegatable fields to the original HVE construction. We use the notation $\sigma \in \Sigma^\ell_{?,\perp}$ to specify a function family. Given TK_σ, one can perform a set of conjunctive equality tests (defined by Equation (4.1)) from the ciphertext. One may also fill in the delegatable fields in σ with any value in $\Sigma \cup \{\perp\}$ and compute a child token for the resulting vector. The child token allows one to evaluate a subset of the functions allowed by the parent token.

Example. The trusted authority T issues to A a token for $\sigma_A = (\mathcal{I}_1, \mathcal{I}_2, ?, ?, \perp, \perp, \ldots, \perp)$. This token allows A to evaluate the following functions from the ciphertext:

- $(x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2)$
- $\forall \mathcal{I}_3 \in \Sigma: \quad (x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2) \wedge (x_3 = \mathcal{I}_3)$

- $\forall \mathcal{I}_4 \in \Sigma: \quad (x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2) \wedge (x_4 = \mathcal{I}_4)$
- $\forall \mathcal{I}_3, \mathcal{I}_4 \in \Sigma: \quad (x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2) \wedge (x_3 = \mathcal{I}_3) \wedge (x_4 = \mathcal{I}_4)$

Later, A delegates to B the token $\sigma_B = (\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3, ?, \perp, \perp, \ldots, \perp)$, where $\mathcal{I}_3 \in \Sigma$. Note that this allows B to evaluate the following functions:

- $(x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2) \wedge (x_3 = \mathcal{I}_3)$
- $\forall \mathcal{I}_4 \in \Sigma: \quad (x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2) \wedge (x_3 = \mathcal{I}_3) \wedge (x_4 = \mathcal{I}_4)$

Clearly, a token for σ_B releases a subset of information allowed by σ_A. Meanwhile, B is able to further delegate on the x_4 field.

4.2.2 dHVE definition

We now give a formal definition of dHVE.

$Setup(1^\lambda)$**.** The $Setup$ algorithm takes as input a security parameter 1^λ and outputs a public key PK and a master secret key MSK.

$Encrypt(\mathsf{PK}, X, \mathbf{Msg})$**.** The $Encrypt$ algorithm takes a public key PK and a pair $(X, \mathbf{Msg}) \in \Sigma^\ell \times \{0,1\}^*$, and outputs a ciphertext $\mathcal{C}$.

$GenToken(\mathsf{PK}, \mathsf{MSK}, \sigma)$**.** The $GenToken$ algorithm takes as input a public key PK, a master secret key MSK, and a vector $\sigma \in (\Sigma_{?,\perp})^\ell$. It outputs a token for evaluating the set of conjunctive queries $\mathcal{C}_\sigma$ from a ciphertext.

$Delegate(\mathsf{PK}, \mathsf{TK}_\sigma, \sigma')$**.** The $Delegate$ algorithm takes as input a public key PK, a token TK_σ for the vector σ, and another vector $\sigma' \prec \sigma$. It outputs a delegated token $\mathsf{TK}_{\sigma'}$ for the new vector σ'.

$Query(\mathsf{PK}, \mathsf{TK}_\sigma, \mathsf{CT}, \sigma')$**.** The $Query$ algorithm takes as input a public key PK, a token TK_σ for the vector σ, a ciphertext CT, and a new vector σ' satisfying the following conditions: (1) $\sigma' \prec \sigma$; (2) σ' does not contain delegatable fields, that is, such a σ' specifies a single conjunctive query (denoted $f_{\sigma'}$) over the point X encrypted. The algorithm outputs $f_{\sigma'}(X)$; if $f_{\sigma'}(X) = 1$, it also outputs the message $\mathbf{Msg}$.

Remark 4.2.2 *In comparison to the general definition given in Section 4.1, in dHVE, we add a payload message* $\mathbf{Msg} \in \{0,1\}^*$ *to the plaintext. Meanwhile, the conjunctive queries in dHVE are functions on the attributes* $X \in \Sigma^\ell$*, but not the payload* $\mathbf{Msg}$*. In addition, if a query matches a point* X *encrypted, one can successfully decrypt the payload message using the corresponding token. It is not hard to show that the above formalization for dHVE is captured by the general definition given in Section 4.1: We can regard* $(\mathbf{Msg}, X)$ *as an entire bit string, and decrypting the payload* $\mathbf{Msg}$ *can be regarded as evaluating a concatenation of bits from the bit string* $(\mathbf{Msg}, X)$*. We choose to define dHVE with a payload message to be consistent with the HVE definition in BW06 [12].*

Selective security of dHVE. We will prove the selective security of our dHVE construction. We give the formal selective security definition below. The full security definition for dHVE can be found in Section 4.7.

- **Init.** The adversary commits to two strings $X_0^*, X_1^* \in \Sigma^\ell$.
- **Setup.** The challenger runs the $Setup$ algorithm and gives the adversary the public key PK.
- **Query 1.** The adversary adaptively makes a polynomial number of "create token", "create delegated token", or "reveal token" queries. The queries must satisfy the following constraint: For any token σ revealed to the adversary, let $\mathcal{C}_\sigma$ denote the set of conjunctive queries corresponding to this token.

$$\forall\ \mathsf{TK}_\sigma \text{ revealed},\ \ \forall f \in \mathcal{C}_\sigma : \quad f(X_0^*) = f(X_1^*) \tag{4.2}$$

- **Challenge.** The adversary outputs two equal-length messages $\mathbf{Msg}_0$ and $\mathbf{Msg}_1$ subject to the following constraint:

  For any token σ revealed to the adversary in the **Query 1** stage, let $\mathcal{C}_\sigma$ denote the set of conjunctive queries corresponding to this token.

$$\forall\ \mathsf{TK}_\sigma \text{ revealed} : \quad \text{if } \exists f \in \mathcal{C}_\sigma, f(X_0^*) = f(X_1^*) = 1, \text{then } \mathbf{Msg}_0 = \mathbf{Msg}_1 \tag{4.3}$$

  The challenger flips a random coin b and returns an encryption of $(\mathbf{Msg}_b, X_b)$ to the adversary.
- **Query 2.** Repeat the **Query 1** stage. All tokens revealed in this stage must satisfy constraints (4.2) and (4.3).
- **Guess.** The adversary outputs a guess b' of b.

The advantage of an adversary $\mathcal{A}$ in the above game is defined to be $\mathsf{Adv}_\mathcal{A} = |\Pr[b = b'] - 1/2|$. We say that a dHVE construction is *selectively secure* if for all polynomial time adversaries, its advantage in the above game is a negligible function of λ.

Observation 4.2.1 *Anonymous Hierarchical Identity-Based Encryption (AHIBE) is a special case of the above-defined dHVE scheme.*

AHIBE is very similar to the dHVE definition given above. The only difference is that in AHIBE, the function family queried is $\mathcal{C}_\sigma$, where σ has the special structure such that $\mathcal{S}(\sigma) = [d]$ where $d \in [\ell]$, $\mathcal{W}(\sigma) = [d+1, \ell]$, and $\mathcal{D}(\sigma) = \emptyset$. In fact, we show that the new security definition and the techniques we use to construct dHVE can be directly applied to give *an AHIBE scheme with shorter private key size*. While the previous AHIBE scheme by Boyen and Waters requires $O(D^2)$ private key size, our new construction has $O(D)$ private key size, where D is the maximum depth of the hierarchy. See Section 4.8 for details of the construction.

4.3 Background on Pairings and Complexity Assumptions

Our construction relies on bilinear groups of composite order $n = pqr$, where p, q, and r are distinct large primes. We now give a background review on bilinear groups of composite order.

Let GG be an algorithm called a *group generator*. Algorithm GG takes as input a security parameter $\lambda \in \mathbb{Z}^{>0}$, a number $k \in \mathbb{Z}^{>0}$, and outputs a tuple $(p, q, r_1, r_2, \ldots, r_k, \mathbb{G}, \mathbb{G}_T, \mathsf{e})$ where $p, q, r_1, r_2, \ldots, r_k$ are $k+2$ distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are two cyclic groups of order $n = pq\prod_{i=1}^{k} r_i$, and e is a function $\mathsf{e} : \mathbb{G}^2 \to \mathbb{G}_T$ satisfying the following properties:

- (Bilinear) $\forall u, v \in \mathbb{G}$, $\forall a, b \in \mathbb{Z}$, $\mathsf{e}(u^a, v^b) = \mathsf{e}(u, v)^{ab}$.
- (Non-degenerate) $\exists g \in \mathbb{G}$ such that $\mathsf{e}(g, g)$ has order n in $\mathbb{G}_T$.

We assume that the group operations in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the bilinear map e are all computable in time polynomial in λ. We also assume that the description of $\mathbb{G}$ and $\mathbb{G}_T$ includes generators of $\mathbb{G}$ and $\mathbb{G}_T$ respectively.

We use the notation $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_{r_1}, \ldots, \mathbb{G}_{r_k}$ to denote the respective subgroups of order p, q, $r_1, \ldots, r_k$ of $\mathbb{G}$. Similarly, we use the notation $\mathbb{G}_{T,p}, \mathbb{G}_{T,q}, \mathbb{G}_{T,r_1}, \ldots, \mathbb{G}_{T,r_k}$ to denote the respective subgroups of order $p, q, r_1, \ldots, r_k$ of $\mathbb{G}_T$.

Our construction relies on two complexity assumptions: the Bilinear Diffie-Hellman assumption (BDH) and the generalized composite 3-party Diffie-Hellman assumption (C3DH). *Although our construction requires only bilinear groups whose order is the product of three primes $n = pqr$, we state our assumptions more generally for bilinear groups of order n where n is the product of three or more primes.*

We begin by defining some notation. We use the notation GG to denote the *group generator* algorithm that takes as input a security parameter $\lambda \in \mathbb{Z}^{>0}$, a number $k \in \mathbb{Z}^{>0}$, and outputs a tuple $(p, q, r_1, r_2, \ldots, r_k, \mathbb{G}, \mathbb{G}_T, \mathsf{e})$ where $p, q, r_1, r_2, \ldots, r_k$ are $k+2$ distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are two cyclic groups of order $n = pq\prod_{i=1}^{k} r_i$, and $\mathsf{e} : \mathbb{G}^2 \to \mathbb{G}_T$ is the bilinear mapping function. We use the notation $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_{r_1}, \ldots, \mathbb{G}_{r_k}$ to denote the respective subgroups of order p, q, $r_1, \ldots, r_k$ of $\mathbb{G}$. Similarly, we use the notation $\mathbb{G}_{T,p}, \mathbb{G}_{T,q}, \mathbb{G}_{T,r_1}, \ldots, \mathbb{G}_{T,r_k}$ to denote the respective subgroups of order $p, q, r_1, \ldots, r_k$ of $\mathbb{G}_T$.

The Bilinear Diffie-Hellman assumption. We review the standard Bilinear Diffie-Hellman assumption, but in groups of composite order. For a given group generator GG define the following distribution $P(\lambda)$:

$$
\begin{array}{l}
(p, q, r_1, \ldots, r_k, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda, k), \quad n \leftarrow pq\prod_{i=1}^{k} r_i, \\
g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad h_1 \xleftarrow{R} \mathbb{G}_{r_1}, \quad \ldots, \quad h_k \xleftarrow{R} \mathbb{G}_{r_k} \\
a, b, c \xleftarrow{R} \mathbb{Z}_n \\
\bar{Z} \leftarrow \big((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}),\ g_q,\ g_p,\ h_1,\ h_2,\ \ldots,\ h_k,\ g_p^a,\ g_p^b,\ g_p^c\big) \\
T \leftarrow \mathsf{e}(g_p,\ g_p)^{abc} \\
\text{Output } (\bar{Z},\ T)
\end{array}
$$

Define algorithm $\mathcal{A}$'s advantage in solving the composite Bilinear Diffie-Hellman problem as

$$\mathsf{cBDH\ Adv}_{\mathsf{GG},\mathcal{A}}(\lambda) := \big|\Pr[\mathcal{A}(\bar{Z}, T) = 1] - \Pr[\mathcal{A}(\bar{Z}, R) = 1]\big|$$

where $(\bar{Z}, T) \xleftarrow{R} P(\lambda)$ and $R \xleftarrow{R} \mathbb{G}_{T,p}$. We say that GG satisfies the composite Bilinear Diffie-Hellman assumption (cBDH) if for any polynomial time algorithm $\mathcal{A}$, $\mathsf{cBDH\ Adv}_{\mathsf{GG},\mathcal{A}}(\lambda)$ is a negligible function of λ.

The generalized composite 3-party Diffie-Hellman assumption. We also rely on the composite 3-party Diffie-Hellman assumption first introduced by Boneh and Waters [12]. For a given group generator GG define the following distribution $P(\lambda)$:

$$(p, q, r_1, \ldots, r_k, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda, k), \quad n \leftarrow pq \prod_{i=1}^{k} r_i,$$
$$g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad h_1 \xleftarrow{R} \mathbb{G}_{r_1}, \quad \ldots, \quad h_k \xleftarrow{R} \mathbb{G}_{r_k}$$
$$R_1, R_2, R_3 \xleftarrow{R} \mathbb{G}_q, \quad a, b, c \xleftarrow{R} \mathbb{Z}_n$$
$$\bar{Z} \leftarrow \big((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}),\ g_q,\ g_p,\ h_1,\ h_2,\ \ldots,\ h_k,\ g_p^a,\ g_p^b,\ g_p^{ab} \cdot R_1,\ g_p^{abc} \cdot R_2\big)$$
$$T \leftarrow g_p^c \cdot R_3$$
$$\text{Output } (\bar{Z},\ T)$$

Define algorithm $\mathcal{A}$'s advantage in solving the generalized composite 3-party Diffie-Hellman problem for GG as $\mathsf{C3DH}\,\mathsf{Adv}_{\mathsf{GG},\mathcal{A}}(\lambda) := \big|\Pr[\mathcal{A}(\bar{Z}, T) = 1] - \Pr[\mathcal{A}(\bar{Z}, R) = 1]\big|$, where $(\bar{Z}, T) \xleftarrow{R} P(\lambda)$ and $R \xleftarrow{R} \mathbb{G}$. We say that GG satisfies the composite 3-party Diffie-Hellman assumption (C3DH) if for any polynomial time algorithm $\mathcal{A}$, its advantage $\mathsf{C3DH}\,\mathsf{Adv}_{\mathsf{GG},\mathcal{A}}(\lambda)$ is a negligible function of λ.

The assumption is formed around the intuition that it is hard to test for Diffie-Hellman tuples in the subgroup $\mathbb{G}_p$ if the elements have a random $\mathbb{G}_q$ subgroup component.

Remark 4.3.1 *Consider bilinear groups of order $n = pqr$, where p, q, and r are three distinct primes. In the above generalized composite 3-party Diffie-Hellman assumption, whether to call a prime p, q, or r is merely a nominal issue. So equivalently, we may assume that it is hard to test for Diffie-Hellman tuples in the subgroup $\mathbb{G}_p$, if each element is multiplied by a random element from $\mathbb{G}_r$ instead of $\mathbb{G}_q$.*

4.4 dHVE Construction

We construct our dHVE scheme by extending the HVE construction by Boneh and Waters [12] (also referred to as the BW06 scheme). One of the challenges that we must overcome is how to add delegation in anonymous IBE systems.

Our primary challenges arise from providing delegation in the anonymous setting. Delegation is easier in non-anonymous IBE systems, such as in HIBE [4]. In the HIBE construction [4], the public key contains an element corresponding to each attribute, and the delegation algorithm can use these elements in the public key to rerandomize the tokens. In anonymous systems, however, as the encryption now has to hide the attributes as well, we have extra constraints on what information we can release in the public key. This restriction on rerandomizing components is the primary hurdle we must overcome.

4.4.1 Construction

In our construction, the public key and the ciphertext are constructed in a way similar to the BW06 scheme. However, we use a new technique to reduce the number of group elements in the ciphertext

asymptotically by one half. Our token consists of two parts, a decryption key part denoted DK and a delegation component denoted DL. The decryption key part DK is similar to that in the BW06 scheme. The delegation component DL is more difficult to construct, since we need to make sure that the delegation component itself does not leak unintended information about the plaintext encrypted.

We will use $\Sigma = \mathbb{Z}_m$ for some integer m. Recall that $\Sigma_{?,\perp} := \Sigma \cup \{?, \perp\}$, where ? denotes a delegatable field, and $\perp$ denotes a "don't care" field.

Setup(1^λ) The setup algorithm first chooses random large primes $p, q, r > m$ and creates a bilinear group $\mathbb{G}$ of composite order $n = pqr$, as specified in Section 4.3. Next, it picks random elements

$$(u_1, h_1), \ldots, (u_\ell, h_\ell) \in \mathbb{G}_p^2 \,, \quad g, v, w, \overline{w} \in \mathbb{G}_p \,, \quad g_q \in \mathbb{G}_q, \quad g_r \in \mathbb{G}_r$$

and an exponent $\alpha \in \mathbb{Z}_p$. It keeps all these as the secret key MSK.

It then chooses $2\ell + 3$ random blinding factors in $\mathbb{G}_q$:

$$(R_{u,1}, R_{h,1}), \ldots, (R_{u,\ell}, R_{h,\ell}) \in \mathbb{G}_q \text{ and } R_v, R_w, \overline{R}_w \in \mathbb{G}_q.$$

For the public key, PK, it publishes the description of the group $\mathbb{G}$ and the values

$$g_q,\ g_r,\ V = vR_v,\ W = wR_w,\ \overline{W} = \overline{w}\overline{R}_w,\ A = \mathsf{e}(g,v)^\alpha,\ \begin{pmatrix} U_1 = u_1 R_{u,1},\ \ H_1 = h_1 R_{h,1} \\ \cdots \\ U_\ell = u_\ell R_{u,\ell},\ \ H_\ell = h_\ell R_{h,\ell} \end{pmatrix}$$

The message space $\mathcal{M}$ is set to be a subset of $\mathbb{G}_T$ of size less than $n^{1/4}$.

Encrypt$(\mathsf{PK},\ X \in \Sigma^\ell,\ \mathbf{Msg} \in \mathcal{M} \subseteq \mathbb{G}_T)$ Assume that $\Sigma \subseteq \mathbb{Z}_m$. Let $X = (x_1, \ldots, x_\ell) \in \mathbb{Z}_m^\ell$. The encryption algorithm first chooses a random $\rho \in \mathbb{Z}_n$ and random $Z,\ Z_0,\ Z_\phi,\ Z_1, Z_2, \ldots, Z_\ell \in \mathbb{G}_q$. (The algorithm picks random elements in $\mathbb{G}_q$ by raising g_q to random exponents from $\mathbb{Z}_n$.) Then, the encryption algorithm outputs the ciphertext:

$$\mathsf{CT} = \left(\widetilde{C} = \mathbf{Msg} A^\rho,\ C = V^\rho Z,\ C_0 = W^\rho Z_0,\ C_\phi = \overline{W}^\rho Z_\phi,\ \begin{pmatrix} C_1 = (U_1^{x_1} H_1)^\rho Z_1, \\ C_2 = (U_2^{x_2} H_2)^\rho Z_2, \\ \cdots\cdots \\ C_\ell = (U_\ell^{x_\ell} H_\ell)^\rho Z_\ell \end{pmatrix} \right)$$

Remark 4.4.1 *We note that the ciphertext size is cut down by roughly a half when compared to the BW06 construction [12]. Therefore, our construction immediately implies an HVE scheme with asymptotically half the ciphertext size as the original BW06 construction.*

GenToken$(\mathsf{PK}, \mathsf{MSK},\ \sigma \in \Sigma_{?,\perp}^\ell)$ The token generation algorithm will take as input the master secret key MSK and an ℓ-tuple $\sigma = (\sigma_1, \ldots, \sigma_\ell) \in \Sigma_{?,\perp}^\ell$. The token for σ consists of two parts: (1) a decryption key component denoted DK, and (2) a delegation component denoted DL.

- The decryption key component DK is composed in a way similar to that of the original HVE construction [12]. Recall that $\mathcal{S}(\sigma)$ denotes the indices of the fixed fields, i.e., indices j such that $\sigma_j \in \Sigma$. Randomly select $\gamma, \overline{\gamma} \in \mathbb{Z}_p$ and $t_j \in \mathbb{Z}_p$ for all $j \in \mathcal{S}(\sigma)$. Pick random $Y, Y_0, Y_\phi \in \mathbb{G}_r$ and $Y_j \in \mathbb{G}_r$ for all $j \in \mathcal{S}(\sigma)$. Observe that picking random elements from the subgroup $\mathbb{G}_r$ can be done by raising g_r to random exponents in $\mathbb{Z}_n$. Next, output the following decryption key component:

$$\mathsf{DK} = \left(K = g^\alpha w^\gamma \overline{w}^{\overline{\gamma}} \textstyle\prod_{j \in \mathcal{S}(\sigma)} (u_j^{\sigma_j} h_j)^{t_j} Y, \quad K_0 = v^\gamma Y_0, \quad K_\phi = v^{\overline{\gamma}} Y_\phi, \quad \forall j \in \mathcal{S}(\sigma) : \; K_j = v^{t_j} Y_j\right)$$

- The delegation component DL is constructed as below. Recall that $\mathcal{W}(\sigma)$ denotes the set of all indices i where $\sigma_i = ?$. Randomly select $Y_{i,u}, Y_{i,h} \in \mathbb{G}_r$. For each $i \in \mathcal{W}(\sigma)$, for each $j \in \mathcal{S}(\sigma) \cup \{i\}$, randomly select $s_{i,j} \in \mathbb{Z}_p$, $Y_{i,j} \in \mathbb{G}_r$. For each $i \in \mathcal{W}(\sigma)$, randomly select $\gamma_i, \overline{\gamma}_i \in \mathbb{Z}_p$, $Y_{i,h}, Y_{i,u}, Y_{i,0}, Y_{i,\phi} \in \mathbb{G}_r$. Next, output the following delegation component DL_i for coordinate i:

$$\forall i \in \mathcal{W}(\sigma) : \; \mathsf{DL}_i = \left(\begin{array}{l} L_{i,h} = h_i^{s_{i,i}} w^{\gamma_i} \overline{w}^{\overline{\gamma_i}} \prod_{j \in \mathcal{S}(\sigma)} (u_j^{\sigma_j} h_j)^{s_{i,j}} Y_{i,h}, \quad L_{i,u} = u_i^{s_{i,i}} Y_{i,u} \\ L_{i,0} = v^{\gamma_i} Y_{i,0}, \; L_{i,\phi} = v^{\overline{\gamma_i}} Y_{i,\phi}, \quad \forall j \in \mathcal{S}(\sigma) \cup \{i\} : \; L_{i,j} = v^{s_{i,j}} Y_{i,j} \end{array} \right)$$

Remark 4.4.2 *Later, if we want to delegate on the k^{th} field by fixing it to $\mathcal{I} \in \Sigma$, we will multiply $L_{k,u}^{\mathcal{I}}$ to $L_{k,h}$, resulting in something similar to the decryption key DK (except without the g^α term). Observe that the $L_{i,h}$ terms encode all the fixed fields (i.e., $\mathcal{S}(\sigma)$). This effectively restricts the use of the delegation components, such that they can only be added on top of the fixed fields, partly ensuring that the delegation components do not leak unintended information.*

$Delegate(\mathsf{PK}, \sigma, \sigma')$ Given a token for $\sigma \in \Sigma_{?,\perp}^{\ell}$, the $Delegate$ algorithm computes a token for $\sigma' \prec \sigma$. Without loss of generality, we assume that σ' fixes only one delegatable field of σ to a symbol in Σ or to $\perp$. Clearly, if we have an algorithm to perform delegation on one field, then we can perform delegation on multiple fields. This can be achieved by fixing the multiple delegatable fields one by one.

We now describe how to compute $\mathsf{TK}_{\sigma'}$ from TK_σ. Suppose σ' fixes the k^{th} coordinate of σ. We consider the following two types of delegation: 1) the k^{th} coordinate is fixed to some value in the alphabet Σ, and 2) the k^{th} coordinate is set to $\perp$, i.e., it becomes a "don't care" field.

Type 1: σ' fixes the k^{th} coordinate of σ to $\mathcal{I} \in \Sigma$, and all other coordinates of σ remain unchanged. In this case, $\mathcal{S}(\sigma') = \mathcal{S}(\sigma) \cup \{k\}$, and $\mathcal{W}(\sigma') = \mathcal{W}(\sigma) \backslash \{k\}$. (Recall that $\mathcal{S}(\sigma)$ denotes the set of indices j where $\sigma_j \in \Sigma$, and $\mathcal{W}(\sigma)$ denotes the set of delegatable fields of σ.)

Step 1: Let $(\mathsf{DK}, \mathsf{DL})$ denote the parent token. Pick a random exponent $\mu \in \mathbb{Z}_n$ and rerandomize the delegation component DL by raising every element in DL to μ. Denote the rerandomized delegation component:

$$\forall i \in \mathcal{W}(\sigma) : \quad \widehat{\mathsf{DL}}_i = \left(\begin{array}{l} \widehat{L}_{i,h} = L_{i,h}^{\mu}, \; \widehat{L}_{i,u} = L_{i,u}^{\mu}, \\ \widehat{L}_{i,0} = L_{i,0}^{\mu}, \; \widehat{L}_{i,\phi} = L_{i,\phi}^{\mu}, \; \forall j \in \mathcal{S}(\sigma) \cup \{i\} : \; \widehat{L}_{i,j} = L_{i,j}^{\mu} \end{array} \right)$$

In addition, compute a partial decryption key component with the k^{th} coordinate fixed to $\mathcal{I}$:

$$\mathsf{pDK} = \left(T = \widehat{L}_{k,u}^{\mathcal{I}}\widehat{L}_{k,h}, \quad T_0 = \widehat{L}_{k,0}, \quad T_\phi = \widehat{L}_{k,\phi}, \quad \forall j \in \mathcal{S}(\sigma') : \;\; T_j = \widehat{L}_{k,j}\right)$$

The partial decryption key pDK is formed similarly to the decryption key DK, except that pDK does not contain the term g^α.

Step 2: Compute $|\mathcal{W}(\sigma')|$ rerandomized versions of the above. For all $i \in \mathcal{W}(\sigma')$, randomly select $\tau_i \in Z_n$, and compute:

$$\mathsf{pDK}_i = \left(\Gamma_i = T^{\tau_i}, \quad \Gamma_{i,0} = T_0^{\tau_i}, \quad \Gamma_{i,\phi} = T_\phi^{\tau_i}, \quad \forall j \in \mathcal{S}(\sigma') : \;\; \Gamma_{i,j} = T_j^{\tau_i}\right)$$

Step 3: Compute the decryption key component DK' of the child token. DK' is computed from two things: 1) DK, the decryption key component of the parent token and 2) pDK, the partial decryption key computed in Step 1. In particular, pDK is the partial decryption key with the k^{th} field fixed; however, as pDK does not contain the g^α term, we need to multiply appropriate components of pDK to those in DK.

To compute DK', first, randomly select $Y', Y'_0, Y'_\phi \in \mathbb{G}_r$. For all $j \in \mathcal{S}(\sigma')$, randomly select $Y'_j \in \mathbb{G}_r$. Now output the following DK':

$$\mathsf{DK}' = \left(\begin{array}{l} K' = KTY', \quad K'_0 = K_0T_0Y'_0, \quad K'_\phi = K_\phi T_\phi Y'_\phi, \quad K'_k = T_kY'_k, \\ \forall j \in \mathcal{S}(\sigma) : \;\; K'_j = K_jT_jY'_j \end{array}\right)$$

Step 4: Compute the delegation component DL' of the child token. DL' is composed of a portion DL'_i for each $i \in \mathcal{W}(\sigma')$. Moreover, each DL'_i is computed from two things: 1) $\widehat{\mathsf{DL}}_i$ as computed in Step 1 and 2) pDK_i as computed in Step 2.

Follow the steps below to compute DL'. For each $i \in \mathcal{W}(\sigma')$, randomly select $Y'_{i,h}, Y'_{i,u}, Y'_{i,0}, Y'_{i,\phi}$ from $\mathbb{G}_r$. For each $i \in \mathcal{W}(\sigma')$, for each $j \in \mathcal{S}(\sigma) \cup \{i, k\}$, pick at random $Y'_{i,j}$ from $\mathbb{G}_r$. Compute the delegation component DL' of the child token:

$$\forall i \in \mathcal{W}(\sigma') : \;\; \mathsf{DL}'_i = \left(\begin{array}{l} L'_{i,h} = \widehat{L}_{i,h}\Gamma_i Y'_{i,h}, \quad L'_{i,u} = \widehat{L}_{i,u}Y'_{i,u}, \\ L'_{i,0} = \widehat{L}_{i,0}\Gamma_{i,0}Y'_{i,0}, \quad L'_{i,\phi} = \widehat{L}_{i,\phi}\Gamma_{i,\phi}Y'_{i,\phi}, \\ L'_{i,i} = \widehat{L}_{i,i}Y'_{i,i}, \quad L'_{i,k} = \Gamma_{i,k}Y'_{i,k}, \quad \forall j \in \mathcal{S}(\sigma) : \;\; L'_{i,j} = \widehat{L}_{i,j}\Gamma_{i,j}Y'_{i,j} \end{array}\right)$$

Type 2: In Type 2 delegation, σ' fixes the k^{th} coordinate of σ to $\perp$. In this case, $\mathcal{S}(\sigma') = \mathcal{S}(\sigma)$, and $\mathcal{W}(\sigma') = \mathcal{W}(\sigma)\backslash\{k\}$. The child token is formed by removing the part DL_k from the parent token:

$$\mathsf{TK}_{\sigma'} = (\mathsf{DK}, \;\; \mathsf{DL}\backslash\{\mathsf{DL}_k\})$$

Remark 4.4.3 *It is not hard to verify that delegated tokens have the correct form, except that their exponents are no longer distributed independently at random, but are correlated*

with the parent tokens. In the proof in Section 4.6, we show that Type 1 delegated tokens "appear" (in a computational sense) as if they were generated directly by calling the $GenToken$ algorithm, that is, with exponents completely at random. This constitutes an important idea in our security proof.

$Query(\mathsf{PK}, \mathsf{TK}_\sigma, \mathsf{CT}, \sigma')$ A token for $\sigma \in \Sigma_{?,\perp}^{\ell}$ allows one to evaluate a set of functions $\mathcal{C}_\sigma$ defined by Equation (4.1) from the ciphertext. Let $\sigma' \prec \sigma$ and assume σ' has no delegatable fields. Then σ' represents a single function $f_{\sigma'}$ (a conjunctive equality test), and the $Query$ algorithm allows us to evaluate $f_{\sigma'}$ over the ciphertext.

To evaluate $f_{\sigma'}$ from the ciphertext using TK_σ, first call the $Delegate$ algorithm to compute a decryption key for σ'. Write this decryption key in the form $\mathsf{DK} = (K, K_0, K_\phi, \forall j \in \mathcal{S}(\sigma') : K_j)$. Furthermore, parse the ciphertext as $\mathsf{CT} = \left(\widetilde{C},\ C,\ C_0,\ C_\phi,\ \forall j \in \ell : C_j\right)$.

Use the same algorithm as the original HVE construction to perform the query. First, compute

$$\mathbf{Msg} \leftarrow \widetilde{C} \cdot \mathsf{e}(C, K)^{-1} \cdot \mathsf{e}(C_0, K_0)\mathsf{e}(C_\phi, K_\phi) \prod_{j \in \mathcal{S}(\sigma')} \mathsf{e}(C_j, K_j) \tag{4.4}$$

If $\mathbf{Msg} \notin \mathcal{M}$, output 0, indicating that $f_{\sigma'}$ is not satisfied. Otherwise, output 1, indicating that $f_{\sigma'}$ is satisfied and also output $\mathbf{Msg}$. We explain why the $Query$ algorithm is correct in Section 4.5.

4.4.2 Security of our construction

Theorem 4.4.1 *Assuming that the Bilinear Diffie-Hellman assumption and the generalized composite 3-party Diffie-Hellman assumptions hold in $\mathbb{G}$, then the above dHVE construction is selectively secure.*

We explain the main techniques used in the proof; however, we defer the detailed proof to Section 4.6. In our main construction, delegated tokens have certain correlations with their parent tokens. As a result, the distribution of delegated tokens differs from tokens generated freshly at random by calling the $GenToken$ algorithm. A major technique used in the proof is "*token indistinguishability*": although delegated tokens have correlations with their parent tokens, they are in fact computationally indistinguishable from tokens freshly generated through the $GenToken$ algorithm. (Strictly speaking, Type 1 delegated tokens are computationally indistinguishable from freshly generated tokens.) This greatly simplifies our simulation, since now the simulator can pretend that all Type 1 tokens queried by the adversary are freshly generated, without having to worry about their correlation with parent tokens. Intuitively, the above notion of token indistinguishability relies on the C3DH assumption: if we use a random hiding factor from $\mathbb{G}_r$ to randomize each term in the token, then DDH becomes hard for the subgroup $\mathbb{G}_p$.

4.5 Correctness

We explain why the $Query$ algorithm is correct. Let $(\mathbf{Msg}, X)$ denote the plaintext encrypted, and let σ' denote the conjunctive query being evaluated in the $Query$ algorithm.

- If the plaintext X satisfies the query, i.e., if $f_{\sigma'}(X) = 1$, a simple calculation shows that the $Query$ algorithm outputs the message $\mathbf{Msg}$. The calculation relies on the fact that if $a \in \mathbb{G}_q$ and $b \in \mathbb{G}_r$, then $\mathsf{e}(a, b) = 1$. Observe that in our construction, each term in the ciphertext (except $\widetilde{C}$) contains a random hiding factor from the subgroup $\mathbb{G}_q$, and each term in the token contains a random hiding factor from the subgroup $\mathbb{G}_r$. When one performs a pairing operation on the ciphertext and the token, the subgroups $\mathbb{G}_q$ and $\mathbb{G}_r$ "disappear", and the result of the pairing is an element of $\mathbb{G}_{T,p}$.
- If the plaintext X does not satisfy the query, i.e., if $f_{\sigma'}(X) = 0$, due to an argument similar to the BW06 [12] paper, the probability $\Pr[Query((\mathsf{PK}, \mathsf{TK}_\sigma, \mathsf{CT}, \sigma') \neq 0]$ is negligible. See Lemma 5.2 of BW06 for details.

4.6 Proof

We prove the security of our construction. We prove selective security, where the adversary commits to two strings X_0^* and X_1^* at the beginning of the security game.

The challenge in proving security is that under our new security game, the simulation needs to reflect how tokens are delegated. In other words, delegated tokens are correlated with their parent tokens in some way, and the simulation should reflect this fact.

Our overall strategy is for the simulator to generate tokens by calling the original $GenToken$ algorithm whenever possible, even when the token is delegated. More specifically, for all Type 1 delegation queries, the simulator generates a freshly randomized token by calling the $GenToken$ algorithm, rather than the $Delegate$ algorithm. As we mentioned, this simulation does not reflect the real security game, since the Type 1 delegated tokens are no longer correlated with their parent tokens. However, we overcome this by showing that the simulation is computationally indistinguishable from the real security game. Intuitively, the indistinguishability property comes from the random group element from the third subgroup $\mathbb{G}_r$ that we use to rerandomize the tokens. Our technique is novel in the sense that in proving semantic security over the ciphertext, we actually rely on "semantic security" over the tokens.

4.6.1 Sequence of games

To prove security, we define a sequence of games, $\mathsf{Game}_0, \mathsf{Game}_1, \ldots, \mathsf{Game}_5$.

Game_0. Let Game_0 denote the real selective security game as defined in Section 4.1.2.

Game_1. We first modify Game_0 slightly into a new game Game_1. Game_1 is almost identical to Game_0, except in the way the tokens are generated. In Game_1, whenever the adversary issues a "create delegated token" query, depending on which type of delegation query it is, the challenger performs the following:

- **Type 1:** The challenger calls the $GenToken$ algorithm to generate a fresh token, and gives it to the adversary.

- **Type 2:** The challenger generates the token in the normal way by calling the $Delegate$ algorithm.

Remark 4.6.1 *The difference between* Game_0 *and* Game_1 *lies in the fact that in the real game* Game_0*, child tokens are always correlated with their parent tokens. In game* Game_1*, a Type 1 delegated token is no longer correlated with its parent token; however, Type 2 delegated tokens are still correlated with their parent tokens.*

Intuitively, if we use the $\mathbb{G}_r$ subgroup to randomize the tokens, no polynomially bounded adversary is able to tell Game_0 apart from Game_1. In other words, the advantage of the adversary in winning Game_0 is almost the same as her advantage in winning Game_1. Therefore, it suffices to prove security using Game_1 instead of Game_0. This simplifies the proof, since in Game_1, Type 1 delegated tokens are formed in the same way as non-delegated tokens.

Lemma 4.6.1 *Assuming that the generalized 3-party Diffie-Hellman assumption holds in* $\mathbb{G}$*, then no polynomially bounded adversary can successfully distinguish* Game_0 *and* Game_1 *with more than negligible advantage.*

Game_2**.** Next, we modify Game_1 slightly into a new game Game_2. Game_2 differs from Game_1 also in the way tokens are formed. To explain how Game_2 differs from Game_1, first observe that any token σ queried must satisfy one of the following two cases:

- *Matching tokens.* The decryption key part of TK_σ matches both of the two selected points X_0^* and X_1^*. In this case, for all $i \in \mathcal{W}(\sigma)$, $X_{0,i}^* = X_{1,i}^*$, since otherwise TK_σ would separate the two selected points. In this case, we say that the token matches both selected points.
- *Non-matching tokens.* The decryption key part of TK_σ matches neither of the two selected points X_0^* and X_1^*.

In Game_2, in any Type 1 delegation query, if the token requested matches both of the selected points X_0^* and X_1^*, the challenger picks the two exponents for w and $\overline{w}$ in DK not independently at random, but in a correlated way: At the beginning of the security game, the challenger picks a random $\pi \in \mathbb{Z}_p$, and keeps it secret from the adversary. Now if a token σ requested in a Type 1 delegation query matches both of the selected points, the challenger picks $\overline{\gamma} = \pi\gamma$ when it computes DK. Similarly, for all $i \in \mathcal{W}(\sigma)$, when the challenger computes DL_i, it picks $\overline{\gamma}_i = \pi\gamma_i$, instead of picking the two exponents independently at random.

Lemma 4.6.2 *Assume that the C3DH assumption holds in* $\mathbb{G}$*, Then for any polynomial time adversary, the difference of advantage in winning* Game_1 *and* Game_2 *is negligible.*

Remark 4.6.2 *In* Game_1*, all tokens (except Type 2 tokens) are picked independently at random. In* Game_2*, this is no longer true, in the sense that for certain queries, the exponents of* w *and* $\overline{w}$ *are correlated with each other. Because of the third subgroup* $\mathbb{G}_r$ *that we use to rerandomize the tokens, we will show that this correlation is computationally hidden from the adversary. The motivation for introducing* Game_2 *is that later the simulator will need to exploit this correlation in* γ *and* $\overline{\gamma}$ *in order to successfully perform a simulation.*

Game_3**.** We now further modify Game_2 into Game_3. Game_3 is almost identical to Game_2 except in the challenge ciphertext. In Game_3, if $\mathbf{Msg}_0 \neq \mathbf{Msg}_1$, the first term $\widetilde{C}$ in the challenge ciphertext

is replaced by a random element from $\mathbb{G}_T$, and the rest of the ciphertext is generated as usual. If $\mathbf{Msg}_0 = \mathbf{Msg}_1$, the challenge ciphertext is generated correctly.

Lemma 4.6.3 *Assume that the BDH and C3DH assumptions hold in $\mathbb{G}$. Then no polynomial time adversary can successfully distinguish* Game_2 *and* Game_3 *with more than negligible probability.*

Game_4**.** Next, we modify Game_3 into a new game Game_4. Game_3 and Game_4 are identical except in the challenge ciphertext. In Game_4, the simulator creates the challenge ciphertext according to the following distribution:

$$C_0 = W^{\rho} g_p^{-\pi\rho'} Z_0, \quad C_\phi = \overline{W}^{\rho} g_p^{\rho'} Z_\phi$$

where ρ' is picked at random from $\mathbb{Z}_p$.

Lemma 4.6.4 *Assume that the C3DH assumption holds in $\mathbb{G}$, Then no polynomial time adversary can successfully distinguish games* Game_3 *and* Game_4 *with more than negligible probability.*

Game_5**.** Let $\overline{E}$ denote the set of indices i such that $X^*_{0,i} \neq X^*_{1,i}$, where X^*_0 and X^*_1 are the two committed points in the selective security game. We now define a new game Game_5. Game_5 differs from Game_4 in that for all $i \in \overline{E}$, the ciphertext component C_i is replaced by a random element from $\mathbb{G}_{pq}$.

Lemma 4.6.5 *Assume that the C3DH assumption holds in $\mathbb{G}$, Then no polynomial time adversary can successfully distinguish* Game_4 *and* Game_5 *with more than negligible probability.*

Notice that in Game_5, the ciphertext gives no information about the point X^*_b or the message $\mathbf{Msg}_b$ encrypted. Therefore, the adversary can win Game_5 with probability at most $1/2$.

We prove the above lemmas. First, we observe that from Game_0 to Game_2, the simulation changes in the way the tokens are generated. We show that these changes remain computationally hidden from any poly-time adversary.

4.6.2 Indistinguishability of Game_0 and Game_1

We prove Lemma 4.6.1 and show that games Game_0 and Game_1 are computationally indistinguishable. To do this, we perform a hybrid argument on the number of Type 1 "Create delegated token" queries issued by the adversary, henceforth referred to as *T*1*-delegation query* for short.

Definition 4.6.6 *Let* $\mathsf{Game}_{0,0} := \mathsf{Game}_0$ *denote the real game. Let* q *denote the number of T*1*-delegation queries issued by the adversary. Define a sequence of hybrid games* $\mathsf{Game}_{0,i}$ *for all* $1 \leq i \leq \mathsf{q}$. $\mathsf{Game}_{0,i}$ *differs from* Game_0 *in the fact that when the adversary issues the first i T*1*-delegation queries, instead of generating the delegated tokens faithfully using the $Delegate$ algorithm, the challenger calls the $GenToken$ algorithm instead to generate these delegated tokens. For all the remaining queries, the challenger computes tokens and responds faithfully as in the real game* Game_0*. Under the above definition,* $\mathsf{Game}_{0,\mathsf{q}}$ *is the same as* Game_1.

Claim 4.6.7 *For all $0 \leq d \leq \mathsf{q} - 1$, no polynomially bounded adversary can distinguish* $\mathsf{Game}_{0,d}$ *from* $\mathsf{Game}_{0,d+1}$ *with more than negligible advantage.*

If we can prove the above Claim 4.6.7, then Lemma 4.6.1 follows by the hybrid argument.

We focus on proving Claim 4.6.7. Intuitively, Claim 4.6.7 relies on the following observation. Pick $h_1, h_2, \ldots, h_\ell \stackrel{R}{\leftarrow} \mathbb{G}_p$, an exponent $\tau \stackrel{R}{\leftarrow} \mathbb{Z}_p$, and randomizing factors Y_1, Y_2, ..., Y_ℓ, Z_1, Z_2, ..., $Z_\ell \stackrel{R}{\leftarrow} \mathbb{G}_r$. Now the tuple

$$(h_1 Z_1, \ldots, h_\ell Z_\ell,\ h_1^\tau Y_1, \ldots, h_\ell^\tau Y_\ell)$$

is computationally indistinguishable from

$$(h_1 Z_1, \ldots, h_\ell Z_\ell, R_1, \ldots, R_\ell),$$

where $(R_1, \ldots, R_\ell)$ are picked independently at random from $\mathbb{G}_{pr} = \mathbb{G}_p \times \mathbb{G}_r$. It is not hard to see that this is the equivalent of the Decisional Diffie-Hellman (DDH) assumption for bilinear groups of composite order. Since we can compute pairing in such groups, normally DDH is easy in group $\mathbb{G}$. However, if we use subgroup $\mathbb{G}_r$ to hide subgroup $\mathbb{G}_p$, DDH becomes hard in $\mathbb{G}_p$. For this reason, we can rerandomize tokens by raising all elements to the same exponent τ, and the rerandomized token is computationally indistinguishable from a completely rerandomized token.

We formalize the above intuition into the ℓ-composite 3-party Diffie-Hellman assumption (ℓ-C3DH). Lemma 4.6.8 proves that the ℓ-C3DH assumption is implied by the generalized C3DH assumption. Therefore, we are not introducing a new assumption here.

Given a group generator GG, define the following distribution $P(\lambda)$:

$$\begin{aligned}
&(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \stackrel{R}{\leftarrow} \mathsf{GG}(\lambda, 1), \quad n \leftarrow pqr,\\
&g_p \stackrel{R}{\leftarrow} \mathbb{G}_p, \quad g_q \stackrel{R}{\leftarrow} \mathbb{G}_p, \quad g_r \stackrel{R}{\leftarrow} \mathbb{G}_r\\
&Y_1, Y_2, \ldots, Y_\ell,\ Z_1, Z_2, \ldots, Z_\ell \stackrel{R}{\leftarrow} \mathbb{G}_r\\
&h_1, h_2, \ldots, h_\ell \stackrel{R}{\leftarrow} \mathbb{G}_p\\
&\tau \stackrel{R}{\leftarrow} \mathbb{Z}_p\\
&X \leftarrow ((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}),\ g_p,\ g_q,\ g_r,\ h_1 Z_1,\ h_2 Z_2,\ \ldots,\ h_\ell Z_\ell)\\
&Q \leftarrow (h_1^\tau Y_1,\ h_2^\tau Y_2,\ \ldots,\ h_\ell^\tau Y_\ell)\\
&\text{Output } (X, Q)
\end{aligned}$$

For an algorithm $\mathcal{A}$, define $\mathcal{A}$'s advantage in solving the above problem:

$$\ell\mathsf{C3DH}\ \mathsf{Adv}_{\mathsf{GG},\mathcal{A}}(\lambda) := \left| \Pr[\mathcal{A}(X, Q) = 1] - \Pr[\mathcal{A}(X, R) = 1] \right|$$

where $(X, Q) \leftarrow P(\lambda)$, and

$$R = (R_1, R_2, \ldots, R_\ell) \stackrel{R}{\leftarrow} \mathbb{G}_{pr}^\ell$$

Lemma 4.6.8 (ℓ-composite 3-party Diffie-Hellman) *Assume that the generalized composite 3-party Diffie-Hellman assumption holds in $\mathbb{G}$. All probabilistic polynomial time adversaries have negligible advantage in solving the ℓ-C3DH problem.*

Proof: By hybrid argument. ■

Proof of Claim 4.6.7: Recall that in game $\mathsf{Game}_{0,d}$, when the challenger receives the first d T1-delegation queries, it creates a completely randomized token. We show that no polynomially bounded adversary has more than negligible advantage in distinguishing $\mathsf{Game}_{0,d}$ from $\mathsf{Game}_{0,d+1}$.

We use the following sequence of games to prove Claim 4.6.7.

$\mathsf{Game}'_{0,d}$ In *Step 1* of the $Delegate$ algorithm, for the $d+1^{\text{th}}$ T1-delegation query, instead of generating $\widehat{\mathsf{DL}} = [\widehat{\mathsf{DL}}_i]_{i \in \mathcal{W}(\sigma)}$ faithfully by raising every element in DL to a random exponent μ, the challenger picks $\widehat{\mathsf{DL}}$ to be a fresh random delegation component. We show that a polynomial time adversary cannot distinguish between the two cases.

$\mathsf{Game}''_{0,d}$ In *Step 2*, instead of computing each pDK_i faithfully, the challenger picks them as fresh random decryption keys (except without the g^α term). We show that a polynomial time adversary cannot distinguish between these two cases.

It is not hard to see that if $\widehat{\mathsf{DL}}$ were a completely rerandomized delegation component for σ, while each pDK_i were independently rerandomized decryption keys (except without the g^α part), then the delegated token $\mathsf{TK}_{\sigma'}$ would be a truly rerandomized token, as if it were generated by directly calling the $GenToken$ algorithm. In other words, $\mathsf{Game}''_{0,d} = \mathsf{Game}_{0,d+1}$. We show below that $\mathsf{Game}_{0,d}$ is indistinguishable from $\mathsf{Game}'_{0,d}$ and that $\mathsf{Game}'_{0,d}$ is indistinguishable from $\mathsf{Game}''_{0,d}$.

■

$\mathsf{Game}_{0,d}$ is indistinguishable from $\mathsf{Game}'_{0,d}$. We prove the above *Step* 1, i.e., $\mathsf{Game}'_{0,d}$ is computationally indistinguishable from $\mathsf{Game}_{0,d}$. Suppose a polynomial time adversary $\mathcal{A}$ can successfully distinguish between the above two games. Let q_0 denote the maximum number of "create token" and "create Type 1 delegated token" queries made by the adversary. We build a simulator $\mathcal{B}$ that leverages $\mathcal{A}$ to break the following $((\ell+1)(\ell+2)\mathsf{q}_0)$-C3DH assumption. We use the notation $\forall i,j,k$ to denote $\forall i \in [\mathsf{q}_0], 0 \le j \le \ell, k \in [\ell+2]$.

$$
\begin{aligned}
&(p,q,r,\mathbb{G},\mathbb{G}_T,\mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda,1), \quad n \leftarrow pqr,\\
&g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_p, \quad g_r \xleftarrow{R} \mathbb{G}_r\\
&\forall i,j,k: \;\; Y_{i,j,k},\; Z_{i,j,k} \xleftarrow{R} \mathbb{G}_r,\;\; v_{i,j,k} \xleftarrow{R} \mathbb{G}_p\\
&\tau \xleftarrow{R} \mathbb{Z}_p\\
&X \leftarrow ((n,\mathbb{G},\mathbb{G}_T,\mathsf{e}),\;\; g_p,\; g_q,\; g_r, \quad \forall i,j,k: \;\; v_{i,j,k}Z_{i,j,k})\\
&Q \leftarrow \left(\forall i,j,k: \;\; v_{i,j,k}^{\tau} Y_{i,j,k}\right)
\end{aligned}
$$

Then the challenger randomly decides to give $(X, Q' = Q)$ or $(X, Q' = R)$, where R is a random vector drawn from $(\mathbb{G}_{pr})^{(\ell+1)(\ell+2)\mathsf{q}_0}$.

The simulator will leverage the adversary $\mathcal{A}$ to distinguish between the above two cases.

Init and Setup. At the beginning of the security game, the adversary commits two points X_0^* and X_1^*.

The simulator picks $v \xleftarrow{R} \mathbb{G}_p$, and for $1 \leq i \leq \ell$, the simulator sets $u_i = v^{x_i}, h_i = v^{y_i}$, where x_i and y_i are random exponents from $\mathbb{Z}_n$. The simulator also picks $w = v^z$ and $\overline{w} = v^{\overline{z}}$. The remaining public parameters and secret key components are picked normally according to the $Setup$ algorithm.

Query 1 and 2. Recall that the adversary makes a number of queries of the following types: 1) create token, 2) create delegated token, 3) reveal token. In this simulation, the simulator computes and saves a token internally whenever a "create token" or "create delegated token" query is made. The simulator simply reveals the saved token whenever the adversary makes a "reveal token" query.

Throughout the simulation, whenever the adversary asks the simulator to create a Type 2 delegated token, the simulator generates it faithfully by deriving it from its parent token. This correctly reflects the relation between the child token and the parent token.

From now on, we focus on how the simulator generates Type 1 delegated tokens and non-delegated tokens.

- Before the adversary issues the $(d+1)^{\text{th}}$ T1-delegation query, the simulator computes tokens using the following strategy. Whenever the adversary asks the simulator to create a Type 1 token or non-delegated token, the simulator incorporates elements from the $(\ell+1)(\ell+2)\mathsf{q}_0$-C3DH instance into these tokens, in a way such that all the exponents are distributed uniformly at random. In particular, let i $(1 \leq i \leq \mathsf{q}_0)$ denote the index of the current query. We note that i is a counter for all "create delegated token" or "create Type 1 delegated token" queries, and d is a counter for all "create Type 1 delegated token" queries. The simulator lets

$$K_0 = v_{i,0,\ell+1} Z_{i,0,\ell+1}, \quad K_\phi = v_{i,0,\ell+2} Z_{i,0,\ell+2}, \quad \forall k \in \mathcal{S}(\sigma): \quad K_k = v_{i,0,k} Z_{i,0,k}$$

For all $j \in \mathcal{W}(\sigma)$, the simulator lets

$$L_{j,0} = v_{i,j,\ell+1} Z_{i,j,\ell+1}, \quad L_{j,\phi} = v_{i,j,\ell+2} Z_{i,j,\ell+2}, \quad \forall k \in \mathcal{S}(\sigma) \cup \{j\}: \quad L_{j,k} = v_{i,j,k} Z_{i,j,k}$$

As the simulator knows the dlog of $w, \overline{w}, u_1, \ldots, u_\ell, h_1, \ldots, h_\ell$ base v, the remaining components of the token can be generated efficiently:

$$K = g^\alpha K_0^z K_\phi^{\overline{z}} \left(\prod_{j \in \mathcal{S}(\sigma)} K_j^{x_j \sigma_j + y_j} \right) Y, \qquad \text{where } Y \xleftarrow{R} \mathbb{G}_r \tag{4.5}$$

$$\forall j \in \mathcal{W}(\sigma): \quad \begin{array}{l} L_{j,h} = L_{j,j}^{y_j} L_{j,0}^z L_{j,\phi}^{\overline{z}} \left(\prod_{k \in \mathcal{S}(\sigma)} L_{j,k}^{x_k \sigma_k + y_k} \right) Y_{j,h} \\ L_{j,u} = L_{j,j}^{x_j} Y_{j,u} \end{array} \quad \text{where } Y_{j,h}, Y_{j,u} \xleftarrow{R} \mathbb{G}_r \tag{4.6}$$

- The adversary makes the $(d+1)^{\text{th}}$ T1-delegation query. In particular, the adversary specifies a parent token, and asks to fix a delegatable field to some value $\mathcal{I} \in \Sigma$. Assume the parent token was created in the i^{th} query, $1 \leq i \leq \mathsf{q}_0$. When performing

Step 1 of the $Delegate$ algorithm, for all $j \in \mathcal{W}(\sigma)$, the simulator lets

$$\widehat{L}_{j,0} = Q'_{i,j,\ell+1},\ \ \widehat{L}_{j,\phi} = Q'_{i,j,\ell+2},\ \ \forall k \in \mathcal{S}(\sigma) \cup \{j\} :\ \ \widehat{L}_{j,k} = Q'_{i,j,k}$$

Here we use the notation $Q'_{i,j,k}$ to index into the vector Q' from the $((\ell+1)(\ell+2)\mathsf{q}_0)$-C3DH problem. As the simulator knows the dlog of $w, \overline{w}, u_1, \ldots, u_\ell, h_1, \ldots, h_\ell$ base v, the remaining components of the token can be generated efficiently due to Equations (4.5) and (4.6).

- For all the remaining queries, the simulator responds faithfully as in the real game.

Clearly, if $Q' = Q$ in the $((\ell+1)(\ell+2)\mathsf{q}_0)$-C3DH instance, then the above simulation is identically distributed as $\mathsf{Game}_{0,d}$. Otherwise, the above simulation is identically distributed as $\mathsf{Game}'_{0,d}$.

Challenge. The simulator generates the challenge ciphertext as normal.

Guess. If the adversary has ϵ difference in its advantage in $\mathsf{Game}_{0,d}$ and $\mathsf{Game}'_{0,d}$, it is not hard to see that the simulator has a comparable advantage in solving the C3DH instance.

$\mathsf{Game}'_{0,d}$ is indistinguishable from $\mathsf{Game}''_{0,d}$. Similarly, we can show that Step 2 above is also true, i.e., no polynomial time adversary can distinguish between $\mathsf{Game}'_{0,d}$ and $\mathsf{Game}''_{0,d}$ with non-negligible probability. To prove this, we further define a sequence of hybrid games. Suppose that in $\mathsf{Game}'_{0,d,c}$ where $0 \le c \le \mathcal{W}(\sigma')$, the first c pDK_i's are replaced by independent random decryption keys (without the g^α part). We show that a polynomial time adversary cannot distinguish between $\mathsf{Game}'_{0,d,c}$ and $\mathsf{Game}'_{0,d,c+1}$. Then, by the hybrid argument, $\mathsf{Game}'_{0,d}$ and $\mathsf{Game}''_{0,d}$ (which is identically distributed as $\mathsf{Game}_{0,d+1}$) are computationally indistinguishable.

The simulator tries to solve the following ℓ-C3DH instance:

$$\begin{aligned}
&(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda, 1), \quad n \leftarrow pqr,\\
&g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_p, \quad g_r \xleftarrow{R} \mathbb{G}_r\\
&Y_1, Y_2, \ldots, Y_\ell,\ Z_1, Z_2, \ldots, Z_\ell \xleftarrow{R} \mathbb{G}_r\\
&v_1, v_2, \ldots, v_\ell \xleftarrow{R} \mathbb{G}_p\\
&\tau \xleftarrow{R} \mathbb{Z}_p\\
&X \leftarrow ((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}),\ \ g_p,\ g_q,\ g_r,\ v_1 Z_1,\ v_2 Z_2,\ \ldots,\ v_\ell Z_\ell)\\
&Q \leftarrow (v_1^\tau Y_1,\ v_2^\tau Y_2,\ \ldots,\ v_\ell^\tau Y_\ell)
\end{aligned}$$

The simulator tries to distinguish between $(X, Q' = Q)$ and $(X, Q' = R)$, where R is a random vector from $\mathbb{G}_{pr}$. The simulator leverages an adversary $\mathcal{A}$ who can distinguish between $\mathsf{Game}'_{0,d,c}$ and $\mathsf{Game}'_{0,d,c+1}$.

Init and Setup. At the beginning of the game, the simulator sets up public parameters and a secret key by choosing $v \xleftarrow{R} \mathbb{G}_p$. For $1 \le i \le \ell$, the simulator sets $u_i = v^{x_i}, h_i = v^{y_i}$, where x_i and y_i are random exponents from $\mathbb{Z}_n$. The simulator also picks $w = v^z$ and $\overline{w} = v^{\overline{z}}$. The

remaining public parameters and secret key components are picked normally according to the $Setup$ algorithm.

Query 1 and 2. The adversary issues a number of queries to the simulator. Like before, the simulator internally computes and saves a token whenever it receives a "create token" or "create delegated token" query. The simulator simply reveals to the adversary the previously computed token in a "reveal token" query.

The simulator treats Type 2 tokens as a special case. Whenever the adversary asks the simulator to create a Type 2 token, the simulator computes it faithfully by deriving the token from the specified parent. This correctly reflects the relation between the child token and its parent. Henceforth, we focus on how the simulator computes Type 1 delegated tokens and non-delegated tokens.

- Before the adversary makes the $(d+1)^{\text{th}}$ T1-delegation query, the simulator always computes each Type 1 delegated token and non-delegated token freshly at random.
- At the $(d+1)^{\text{th}}$ T1-delegation query, the adversary specifies a parent token, and requests to fix the k^{th} coordinate to some value $\mathcal{I} \in \Sigma$. To answer this query, the simulator first generates $\widehat{\mathsf{DL}}_i$ for all $i \in \mathcal{W}(\sigma)$, and pDK. For $i \in \mathcal{W}(\sigma) \backslash \{k\}$, the simulator picks at random $\widehat{L}_{i,0}, \widehat{L}_{i,\phi}$ and $\widehat{L}_{i,j}$ for all $j \in \mathcal{S}(\sigma) \cup \{i\}$. The simulator lets

$$T_0 = \widehat{L}_{k,0} = v_{\ell+1} Z_{\ell+1}, \quad T_\phi = \widehat{L}_{k,\phi} = v_{\ell+2} Z_{\ell+2}, \quad \forall j \in \mathcal{S}(\sigma') : \ T_j = \widehat{L}_{k,j} = v_j Z_j$$

  As the simulator knows the dlog of $w, \overline{w}, u_1, \ldots, u_\ell, h_1, \ldots, h_\ell$ base v, the remaining components of $\widehat{\mathsf{DL}}_i$'s and pDK can be generated efficiently in a way similar to Equations (4.5) and (4.6). The only difference is that pDK does not contain the g^α term, while a decryption key DK does.

  The simulator picks the first c pDK_i's as fresh random (partial) decryption keys.

  Let i be the $c+1^{\text{th}}$ index in $\mathcal{W}(\sigma')$. For pDK_i, the simulator sets

$$\Gamma_{i,0} = Q'_{\ell+1}, \quad \Gamma_{i,\phi} = Q'_{\ell+2} \qquad \forall j \in \mathcal{S}(\sigma') : \ \Gamma_{i,j} = Q'_j$$

  We use the notation Q'_j to index into the j^{th} element of the vector Q' from the ℓ-C3DH problem. Again, since the simulator knows the dlog of $w, \overline{w}, u_1, \ldots, u_\ell, h_1, \ldots, h_\ell$ base v, the remaining terms in pDK_i can be generated efficiently.

  For all the remaining pDK_i's, the simulator generates them normally as in the original $Delegate$ algorithm.
- For all the remaining queries, the simulator generates them faithfully.

Challenge. The simulator generates the challenge ciphertext as normal.

Guess. Notice that if $Q' = Q$ in the ℓ-C3DH problem, then the above simulation is identically distributed as $\mathsf{Game}_{0,d,c}$; otherwise, the above simulation is identically distributed as $\mathsf{Game}_{0,d,c+1}$. Therefore, if a polynomial time adversary could successfully distinguish between $\mathsf{Game}_{0,d,c}$ and $\mathsf{Game}_{0,d,c+1}$, then the simulator would be able to solve the ℓ-C3DH problem with non-negligible probability.

4.6.3 Indistinguishability of Game_1 and Game_2

We prove Lemma 4.6.2.

Let q denote the maximum number of T1-delegation queries for a matching token made by the adversary. We show that if a poly-time adversary has non-negligible difference in its advantage in Game_1 and Game_2, we can build a simulator that leverages this adversary to break the modified $\mathsf{q}(\ell+1)$-C3DH assumption. In the following, we use $\forall i, j$ to mean $\forall i \in [\mathsf{q}], 0 \leq j \leq \ell$.

$$
\begin{aligned}
&(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda, 1), \quad n \leftarrow pqr, \\
&g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_p, \quad g_r \xleftarrow{R} \mathbb{G}_r \\
&\forall i, j: \;\; Y_{i,j,1}, Y_{i,j,2} \xleftarrow{R} \mathbb{G}_r \\
&v_1, v_2 \xleftarrow{R} \mathbb{G}_p \\
&\forall i, j: \;\; \tau_{i,j} \xleftarrow{R} \mathbb{Z}_p \\
&Q \leftarrow \left(v_1^{\tau_{i,j}} Y_{i,j,1},\; v_2^{\tau_{i,j}} Y_{i,j,2}\right)
\end{aligned}
$$

The modified $\mathsf{q}(\ell+1)$-C3DH assumption says that given randomly $(Q' = Q)$ or $(Q' = R)$ where R is a random vector of length $\mathsf{q}(\ell+1)$ from $\mathbb{G}_{pr}$, a poly-time adversary cannot distinguish whether $Q' = Q$ or $Q' = R$. The modified $\mathsf{q}(\ell+1)$-C3DH assumption follows from the generalized C3DH assumption by the hybrid argument. Hence, we are not introducing a new assumption here.

Suppose the simulator is randomly given $(Q' = Q)$ or $(Q' = R)$ where R is a random vector of length $\mathsf{q}(\ell+1)$ from $\mathbb{G}_{pr}$. Now the simulator tries to distinguish between the two cases.

The simulator first generates public and secret keys. The simulator picks $v \in \mathbb{G}_p$ at random. For $1 \leq i \leq \ell$, the simulator sets $u_i = v^{x_i}, h_i = v^{y_i}$, where x_i and y_i are random exponents from $\mathbb{Z}_n$. The simulator also picks $w = v^z$ and $\overline{w} = v^{\overline{z}}$, where z and $\overline{z}$ are also random exponents from $\mathbb{Z}_n$. The simulator proceeds and generates the rest of public and secret keys as normal.

We explain how the simulator answers the adversary's queries. When the adversary makes the i^{th} T1-delegation query for a matching token, the simulator computes a token by letting $K_0 = Q'_{i,0,1}$ and $K_\phi = Q'_{i,0,2}$. This fixes the exponents γ and $\overline{\gamma}$, although the simulator does not know what γ and $\overline{\gamma}$ really are. The simulator picks the remaining parameters needed as normal, and computes the decryption key part DK. Notice that even though the simulator does not know γ or $\overline{\gamma}$, DK can be efficiently computed, since the simulator knows the dlog of $w, \overline{w}$ base v.

Similarly, for delegation component DL_j where $j \in \mathcal{W}(\sigma)$, the simulator lets $L_{i,0} = Q'_{i,j,1}$, $L_{i,\phi} = Q'_{i,j,2}$, picks the remaining parameters needed as normal, and computes DL_j. By the same reasoning, even though the simulator does not know γ_j or $\overline{\gamma}_j$, DL_j can be efficiently computed since the simulator knows the dlog of $w, \overline{w}$ base v.

We observe that if $Q' = Q$, then the above simulation would be identically distributed as Game_2. Otherwise, if $Q' = R$, the above simulation would be identically distributed as Game_1. Therefore, if a poly-time adversary has non-negligible difference in its advantage in distinguishing Game_1 and Game_2, the simulator would be able to break the modified $\mathsf{q}(\ell+1)$-C3DH assumption.

Until now, we have shown that the simulator can change the way tokens are computed such that these changes remain computationally hidden from the adversary. Now we show that if the simulator changes certain parts of the ciphertext to random, a poly-time adversary cannot distinguish with more than negligible advantage.

In all the simulations described below, the simulator will compute tokens only when a "reveal token" query is made. When the adversary makes a "create token" or "create delegated token" query, the simulator simply records that query without computing the actual token created. In particular, in some of these simulations, the simulator is not able to compute all tokens. However, the simulator is always able to compute a token in a "reveal token" query. Recall that a token σ represents a set of conjunctive queries over the point X encrypted. Any token σ requested in a "reveal token" query must satisfy the condition that for any function $f \in \mathcal{C}_\sigma$ (f is a conjunctive query on $X \in \mathbb{Z}_m^\ell$), $f(X_0^*) = f(X_1^*)$. Henceforth, we use the terminology *σ does not separate the two selected points X_0^* and X_1^** to describe the above condition. In all the simulations below, the simulator is always able to compute a token σ, as long as σ does not separate the two selected points.

In the simulations described below that change certain parts of the ciphertext, an adversary can ask the simulator to reveal a token of the following types: 1) non-delegated, 2) Type 1 delegated, 3) Type 2 delegated. Clearly, non-delegated tokens are distributed independently from other tokens. Due to Lemma 4.6.1, Type 1 tokens appear to be uncorrelated with their parent tokens. Therefore, the simulator always computes non-delegated and Type 1 tokens freshly at random. By contrast, Type 2 tokens are correlated with their ancestor tokens, and thus require special treatment. The simulator must construct Type 2 tokens such that they reflect the correct relationship with their ancestors. Before explaining how the simulations are performed, we describe a general strategy the simulator uses to generate Type 2 delegated tokens, since they require special treatment different from that for non-delegated tokens and Type 1 delegated tokens.

4.6.4 Generating Type 2 delegated tokens

The simulator uses a "book-keeping" technique. We use the notation $\mathsf{TK}_{\sigma'} \prec_2 \mathsf{TK}_\sigma$ to mean that $\mathsf{TK}_{\sigma'}$ is derived from TK_σ through a Type 2 delegation operation. Whenever the adversary asks the simulator to reveal a Type 2 delegated token, instead of computing a fresh token, the simulator examines the history of queries, and finds the sequence of Type 2 delegation queries that created this token,

$$\mathsf{TK}_{\sigma_k} \prec_2 \mathsf{TK}_{\sigma_{k-1}} \prec_2 \ldots \prec_2 \mathsf{TK}_{\sigma_1}$$

where $\mathsf{TK}_{\sigma_k} := \mathsf{TK}_\sigma$ is the currently requested token, and TK_{σ_1} is a non-delegated token or a Type 1 delegated token. We note that the simulator might not be able to compute all these tokens. However, the simulator can compute a token if the token does not separate the two selected points X_0^* and X_1^*.

If a token $\mathsf{TK}_{\sigma_i} (1 \leq i \leq k)$ in the above sequence has been computed by the simulator in the past, the simulator simply derives TK_σ from TK_{σ_i} using the $Delegate$ algorithm, and returns it to the adversary. In particular, σ fixes some delegatable coordinates of σ_i to $\perp$, and the simulator simply removes the corresponding delegation components from TK_{σ_i} to form TK_σ. Otherwise, if

no token in the above sequence has been computed by the simulator in the history, the simulator finds the earliest ancestor $\mathsf{TK}_{\sigma_i} (1 \leq i \leq k)$ in the above sequence, such that TK_{σ_i} does not separate the two selected points X_0^* and X_1^*. The simulator generates TK_{σ_i} freshly at random, and then it follows the $Delegate$ algorithm to generate TK_σ from TK_{σ_i} (by removing the fields set to $\perp$ from the delegation components).

We now describe a sequence of simulations that replace ciphertext components by random group elements. In these simulations, we focus on how the simulator can compute non-delegated and Type 1 tokens. Type 2 tokens are always treated as a special case using the algorithm described earlier in this section.

4.6.5 Indistinguishability of Game_2 and Game_3

In Game_3, if $\mathbf{Msg}_0 \neq \mathbf{Msg}_1$, the challenger replaces the ciphertext component $\widetilde{C}$ by a random group element from $\mathbb{G}_T$.

The proof that Game_2 and Game_3 are indistinguishable to a poly-time adversary is similar to that in the original BW06 paper [12].

We prove this in two steps:

- Game'_2: If $\mathbf{Msg}_0 \neq \mathbf{Msg}_1$, the challenger replaces the ciphertext component $\widetilde{C}$ by a random group element from $\mathbb{G}_{T,p}$. No poly-time adversary can distinguish Game'_2 from Game_2 with more than negligible probability.
- Because of the subgroup decision assumption (implied by the C3DH assumption), if the simulator replaces the ciphertext component $\widetilde{C}$ by a random group element from $\mathbb{G}_T$ instead of $\mathbb{G}_{T,p}$, the adversary cannot distinguish this case from Game'_2.

We first prove that Game_2 is computationally indistinguishable from Game'_2. Suppose the simulator tries to solve the following BDH instance:

$$
\begin{aligned}
&(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda), \quad n \leftarrow pqr, \quad g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad g_r \xleftarrow{R} \mathbb{G}_r \\
&a, b, c \xleftarrow{R} \mathbb{Z}_n \\
&\bar{Z} \leftarrow \big((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}),\ g_p,\ g_q,\ g_r,\ g_p^a,\ g_p^b,\ g_p^c\big) \\
&Q \leftarrow \mathsf{e}(g_p, g_p)^{abc}
\end{aligned}
$$

The simulator is randomly given $(\bar{Z},\ Q' = Q)$ or $(\bar{Z},\ Q' = R)$ where R is a random element in $\mathbb{G}_T$, and it tries to distinguish between these two cases.

If there exists a poly-time adversary $\mathcal{A}$ that has non-negligible difference in its advantage in Game_2 and Game_3, we can build the following simulation to solve the BDH instance.

Init. The adversary commits to two selected points X_0^* and X_1^*. The challenger picks a random coin β internally.

Setup. The simulator chooses random $(R_{u,1}, R_{h,1}), (R_{u,2}, R_{h,2}), \ldots, (R_{u,\ell}, R_{h,\ell}) \in \mathbb{G}_q$, R_v, R_w, $\overline{R}_w \in \mathbb{G}_q$, and random $z_1, y_1, \ldots, t_\ell, y_\ell \in \mathbb{Z}_n$, and $x, \overline{x} \in \mathbb{Z}_n$. The simulator publishes the group description $g_q, g_r, V = g_p R_v$. It lets $A = \mathsf{e}(g_p^a, g_p^b)$ and creates

$$U_i = (g_p^b)^{z_i} R_{u,i}, \quad H_i = (g_p^b)^{-z_i X^*_{\beta,i}} g_p^{y_i} R_{h,i}$$

Finally, the simulator creates:

$$W = g^x, \quad \overline{W} = g^{\overline{x}}$$

We observe that the parameters are distributed identically to the real scheme.

Query 1. The simulator does not compute any token when the adversary makes "create token" or "create delegated token" queries. It computes tokens only when "reveal token" queries are made.

Recall that in Section 4.6.4, we pointed out that Type 2 tokens require special treatment. In addition, we gave an algorithm for the simulator to generate Type 2 tokens such that they reflect the correct relationship with their parent tokens. Now it suffices to show that the simulator can always compute a fresh random token, so long as the token does not separate the two selected points X^*_0 and X^*_1.

Whenever the adversary makes a "reveal token" query for a matching token, the simulator simply aborts and takes a random guess. The reason is that by our definition, when the adversary asks the simulator to reveal a matching token, the challenge messages $\mathbf{Msg}_0$ and $\mathbf{Msg}_1$ must be equal. However, in this case, Game_2 and Game_3 are identical, so there can be no difference in the adversary's advantage in between these two games.

Whenever the adversary asks the simulator to reveal a non-matching token, the simulator needs to compute a token of the correct form. First, notice that the delegation components DL can be efficiently computed, since they do not contain any unknown parameters. However, computing the decryption key component DK is slightly more tricky. Recall that because of the way the public key is formed, $g^\alpha = g_p^{ab}$. Therefore, the decryption key component DK contains the term g_p^{ab}. Unfortunately, the simulator does not know g_p^{ab}, so it has to find some way to cancel out that term and still form a correctly distributed token. The intuition is that since the token is non-matching, there exists a dimension i where $X^*_{0,i} \neq \sigma_i$ and $X^*_{1,i} \neq \sigma_i$. We observe that the term $u_i^{\sigma_i} h_i = (g_p^b)^{\Delta_i z_i} g_p^{y_i}$ contains $(g_p^b)^{\Delta_i z_i}$, where $\Delta_i = \sigma_i - X^*_{\beta,i} \neq 0$. Therefore, the simulator can pick $\widehat{t_i}$ at random from $\mathbb{Z}_n$, and let

$$t_i = \widehat{t_i} - a/(\Delta_i z_i)$$

without actually computing it. And this t_i is used to generate the decryption key component DK. If the simulator picks t_i in the way specified above, it is able to compute DK, since all terms containing the unknown parameter g_p^{ab} cancel out. In particular, in the decryption key DK, K is a product of several terms. Rewrite K:

$$\begin{aligned} K &= g_p^{ab} w^\gamma \overline{w}^{\overline{\gamma}} \prod_{j \in \mathcal{S}(\sigma)} (u_j^{\sigma_j} h_j)^{t_j} Y \\ &= \left(g_p^{ab} (u_i^{\sigma_i} h_i)^{t_i} \right) \cdot \left(w^\gamma \overline{w}^{\overline{\gamma}} \prod_{j \in \mathcal{S}(\sigma), j \neq i} (u_j^{\sigma_j} h_j)^{t_j} Y \right) \end{aligned}$$

The product term

$$g_p^{ab}(u_i^{\sigma_i}h_i)^{t_i} = (u_i^{\sigma_i}h_i)^{\widehat{t}_i} \cdot (g_p^a)^{-y_i/(\Delta_i z_i)}$$

can be efficiently computed, since all terms involving g_p^{ab} cancel out. It is not hard to see that the remaining terms in K can be efficiently generated, since the simulator knows all parameters needed. As the simulator knows g_p^a, the term $K_i = v^{t_i}$ can be efficiently computed.

Challenge. The adversary gives the simulator two messages, $\mathbf{Msg}_0$ and $\mathbf{Msg}_1$. If $\mathbf{Msg}_0 = \mathbf{Msg}_1$, the simulator aborts and takes a random guess for the reason stated above.

Otherwise, the simulator chooses random $Z, Z_0, Z_\phi, Z_1, Z_2, \ldots, Z_\ell \in \mathbb{G}_q$, and outputs the following challenge ciphertext:

$$\widetilde{C} = \mathbf{Msg}_\beta Q', \;\; C = (g_p^c)Z, \;\; C_0 = (g_p^c)^x Z_0, \;\; C_\phi = (g_p^c)^{\overline{x}} Z_\phi, \;\; \forall i \in [\ell] : \; C_i = (g_p^c)^{y_i} Z_i$$

Query 2. Same as phase Query 1.

Guess. The adversary outputs a guess β'. If $\beta = \beta'$, the simulator guesses that $Q' = Q$. Otherwise, the simulator guesses that $Q' = R$. We observe that if $Q' = Q$, the ciphertext component $\widetilde{C}$ is a faithful encryption of $\mathbf{Msg}_\beta$; otherwise, $\widetilde{C}$ is distributed at random in $\mathbb{G}_{T,p}$. Therefore, if the adversary has ϵ advantage in guessing β, the simulator also has ϵ advantage in solving the BDH instance.

To show that Game_2' is computationally indistinguishable from Game_3, we rely on the Bilinear Subgroup Decision (BSD) assumption introduced by Boneh, Sahai and Waters [11]. Bilinear Subgroup Decision assumption is implied by the generalized composite 3-party Diffie-Hellman assumption.

The simulator gets the following BSD instance:

$$(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda), \quad n \leftarrow pqr, \quad g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad g_r \xleftarrow{R} \mathbb{G}_r$$
$$\bar{Z} \leftarrow ((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}), \;\; g_p, \; g_q, \; g_r)$$
$$Q \leftarrow \mathbb{G}_{T,p}$$

The simulator is also randomly given $Q' = Q$ or $Q' = R$ where $R \xleftarrow{R} \mathbb{G}_T$. The BSD assumption posits that no poly-time algorithm can distinguish between the above two cases with more than negligible advantage.

The simulation proceeds as follows.

Init. The attacker gives the simulator two identities X_0^*, X_1^*. The challenger then flips the coin β internally.

Setup. The simulator sets up the parameters as would the real setup algorithm. All the simulator needs to do this is g_p, g_q, g_r from the assumption.

Query 1. The simulator answers queries as the real authority would. One small difference is that the simulator chooses exponents from $\mathbb{Z}_n$ instead of $\mathbb{Z}_p$. However, this does not change anything since the both the simulator and a real authority will raise the elements from $\mathbb{G}_p$ to the exponents.

Challenge. The adversary first gives the simulator messages $\mathbf{Msg}_0, \mathbf{Msg}_1$. If $\mathbf{Msg}_0 = \mathbf{Msg}_1$ then the simulator simply encrypts the message to the point X^*_β. Otherwise, the simulator creates the challenge ciphertext of message $\mathbf{Msg}_\beta$ to X^*_β as normal with the exception that C' is multiplied by Q'.

If $Q' = Q$, then the simulator is playing Game'_2; otherwise it is playing Game_3.

Query 2. Same as Query Phase 1.

Guess. The adversary outputs a guess β'. If $\beta = \beta'$, the simulator guesses that $Q' = Q$; otherwise it guesses that $Q' = R$. By our assumption the probability that the adversary guesses β correctly in Game'_2 has a non-negligible ϵ difference from that of it guessing it correctly in Game_3. However, it is in Game_3 if and only if the challenger gave the simulator $Q' = R$ instead of $Q' = Q$. Therefore, the simulator has advantage ϵ in the Bilinear Subgroup Decision game, implying that the simulator has an advantage of ϵ in the Composite 3-Party Diffie-Hellman game.

4.6.6 Indistinguishability of Game_3 and Game_4

If a polynomial time adversary $\mathcal{A}$ has non-negligible difference ϵ between its advantage in Game_3 and Game_4, we can build a simulator $\mathcal{B}$ that breaks the C3DH assumption with probability ϵ.

The challenger first creates a 3-Party challenge:

$$
\begin{aligned}
&(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda), \quad n \leftarrow pq, \quad g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad g_r \xleftarrow{R} \mathbb{G}_r \\
&R_1, R_2, R_3 \xleftarrow{R} \mathbb{G}_q \\
&a, b, c \xleftarrow{R} \mathbb{Z}_n \\
&\bar{Z} \leftarrow \big((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}),\ g_p,\ g_q, g_r,\ g_p^a,\ g_p^b,\quad \Gamma = g_p^{ab} \cdot R_1,\quad Y = g_p^{abc} \cdot R_2\big) \\
&Q \leftarrow g_p^c \cdot R_3
\end{aligned}
$$

It then randomly decides whether to give $(\bar{Z},\ Q' = Q)$ or $(\bar{Z},\ Q' = R)$ where R is a random element in $\mathbb{G}_{pq}$.

We create the following simulation:

Init. The adversary commits to two points X^*_0 and X^*_1. The simulator flips a random coin β internally.

Setup. The simulator picks $V = g_p R_v$, where R_v is picked at random from $\mathbb{G}_q$. The simulator also picks from $\mathbb{Z}_n$ random exponents $y, x, \overline{x}, \mu_i, z_i$ for each $i \in [\ell]$, and lets

$$W = g_p^y \cdot \Gamma^x, \quad \overline{W} = \Gamma^{\overline{x}}$$

The simulator creates:

$$\forall i \in [\ell] : \;\; U_i = (g_p^b)^{\mu_i} R_{u,i}, \;\; H_i = (g_p^b)^{-\mu_i X^*_{\beta,i}} g_p^{z_i} R_{h,i}$$

where $R_{u,i}$ and $R_{h,i}$'s are random group elements from $\mathbb{G}_q$. The simulator also chooses a random $\alpha \in \mathbb{Z}_n$, and computes $A = \mathsf{e}(g_p, V)^\alpha$.

Query 1. Recall that each query σ defines a set of conjunctive queries $\mathcal{C}_\sigma$ on the encrypted point X. Whenever the adversary asks the simulator to reveal a token for σ, σ must satisfy the condition that for any function $f \in \mathcal{C}_\sigma$ (f is a conjunctive query on $X \in \mathbb{Z}_m^\ell$), $f(X_0^*) = f(X_1^*)$. Henceforth, we use the terminology σ does not separate the two selected points X_0^* and X_1^* to denote the above condition.

We now describe how the simulator responds to the adversary's "reveal token" queries. The token can be non-delegated, Type 1 delegated, or Type 2 delegated. Type 1 delegated tokens and non-delegated tokens should be generated freshly at random, while Type 2 tokens should reflect the correct relation with their parent tokens. In Section 4.6.4, we gave an algorithm for generating Type 2 tokens. Hence, it suffices to show how the simulator can compute fresh random tokens.

- If the token matches both selected points, the simulator first picks a random τ from $\mathbb{Z}_n$, and lets $\gamma = -\overline{x}\tau$, and $\overline{\gamma} = x\tau$. Similarly, the simulator picks a random $\tau_i \in \mathbb{Z}_n$ for each $i \in \mathcal{W}(\sigma)$, and lets $\gamma_i = -\overline{x}\tau_i$, and $\overline{\gamma}_i = x\tau_i$. Except for the above, the simulator follows the $GenToken$ algorithm and computes the token. Notice that the token can be computed efficiently, since the only unknown term involving g_p^{ab} cancels out because of the way the simulator chose $\gamma, \overline{\gamma}$, and the way the simulator chose γ_i and $\overline{\gamma}_i$'s. In particular, consider the term K in the decryption key component DK. Group the terms in K:

$$K = \left(w^\gamma \overline{w}^{\overline{\gamma}}\right)\left(g^\alpha \prod_{j \in \mathcal{S}(\sigma)} (u_j^{\sigma_j} h_j)^{t_j} Y\right)$$

  In the above, the product term $w^\gamma \overline{w}^{\overline{\gamma}}$ can be efficiently computed since all terms involving g_p^{ab} cancel out:

$$w^\gamma \overline{w}^{\overline{\gamma}} = g_p^{-\overline{x}\tau y}$$

  Similarly, for all $i \in \mathcal{W}(\sigma)$, the following term in the delegation component DL_i can be efficiently computed:

$$w^{\gamma_i} \overline{w}^{\overline{\gamma}_i} = g_p^{-\overline{x}\tau_i y}$$

  Clearly, all remaining terms in DK or DL can be efficiently computed, since the simulator knows all necessary parameters.

- If the token matches neither selected point, there exists coordinate $c \in \mathcal{S}(\sigma)$, such that $\Delta_c = \sigma_c - X^*_{\beta,c} \neq 0$. In this case, the simulator uses the following strategy to compute the decryption key component DK. The simulator first picks random $\gamma, \overline{\gamma} \in \mathbb{Z}_n$. It also picks random $\widehat{t_c} \in \mathbb{Z}_n$, and lets

$$t_c = \widehat{t}_c - \frac{a(\gamma x + \overline{\gamma x})}{\mu_c \Delta_c}$$

without actually computing t_c. Except for the above, the simulator follows the $GenToken$ algorithm to compute the token requested. Notice that the token can be computed efficiently, since all terms involving the unknown parameter g_p^{ab} cancel out. In particular, in the decryption key components DK, group the terms in K:

$$K = \left(w^{\gamma} \overline{w}^{\overline{\gamma}} (u_c^{\sigma_c} h_c)^{t_c} \right) \left(g^{\alpha} \prod_{j \in \mathcal{S}(\sigma), j \neq c} (u_j^{\sigma_j} h_j)^{t_j} Y \right)$$

The product term $w^{\gamma} \overline{w}^{\overline{\gamma}} (u_c^{\sigma_c} h_c)^{t_c}$ can be efficiently computed, since all terms involving the unknown parameter g_p^{ab} cancel out:

$$w^{\gamma} \overline{w}^{\overline{\gamma}} (u_c^{\sigma_c} h_c)^{t_c} = g_p^{y\gamma} (u_c^{\sigma_c} h_c)^{\widehat{t}_c} (g_p^a)^{-z_c \Theta_c}$$

where $\Theta_c = \frac{(\gamma x + \overline{\gamma x})}{\mu_c \Delta_c}$. In addition, we observe that the term $K_c = v^{t_c} Y_c$ can be computed efficiently since the simulator knows g_p^a. Clearly, all other terms in DK can be computed efficiently.

To generate the delegation components DL, we can apply the same trick, i.e., by letting

$$s_{i,c} = \widehat{s}_{i,c} - \frac{a(\gamma_i x + \overline{\gamma}_i \overline{x})}{\mu_c \Delta_c}$$

for every $i \in \mathcal{W}(\sigma)$. $\widehat{s}_{i,c}$ is picked at random from $\mathbb{Z}_n$.

Challenge. The adversary submits two messages $\mathbf{Msg}_0$ and $\mathbf{Msg}_1$ to the simulator. The simulator creates the following ciphertext:

$$C = Q', \;\; C_0 = Q'^{y} Y^{x} Z_0, \;\; C_{\phi} = Y^{\overline{x}} Z_{\phi}, \;\; \forall i \in [\ell] : \;\; C_i = Q'^{z_i} Z_i$$

In addition, if $\mathbf{Msg}_0 = \mathbf{Msg}_1$, the simulator lets $\widetilde{C} = \mathsf{e}(g_p, Q')^{\alpha}$. Otherwise, $\widetilde{C}$ is replaced by a random element from $\mathbb{G}_T$. Observe that if $Q' = Q$, the ciphertext is identically distributed as in Game_3. Otherwise, if Q' is a random element from $\mathbb{G}_{pq}$, the ciphertext is identically distributed as in Game_4.

Query 2. Same as the **Query 1** stage.

Guess. The adversary outputs a guess β' of β. By the C3DH assumption, a poly-time adversary cannot have more than negligible difference in its advantage in Game_3 and Game_4.

4.6.7 Indistinguishability of Game_4 and Game_5

Let $\overline{E}$ denote the set of indices i where the two committed points are not equal, i.e., $X^*_{0,i} \neq X^*_{1,i}$. Let $\mathsf{Game}_{4,0} := \mathsf{Game}_4$. We define a sequence of games $\mathsf{Game}_{4,1}, \mathsf{Game}_{4,2}, \ldots, \mathsf{Game}_{4,|\overline{E}|}$. Let $\widetilde{E}_i \subseteq \overline{E}$ denote the first i indices in $\overline{E}$. In $\mathsf{Game}_{4,i}$ ($1 \leq i \leq |\overline{E}|$), the challenger creates ciphertext components $\widetilde{C}, C$, and C_j normally for all $j \notin \widetilde{E}_i$. For all $j \in \widetilde{E}_i$, the challenger replaces C_j with a random group element from $\mathbb{G}_{pq}$. For C_0, C_ϕ, the challenger creates the following ciphertext components like in game Game_4:

$$C_0 = W^\rho g_p^{-\pi\rho'} Z_0, \quad C_\phi = \overline{W}^\rho g_p^{\rho'} Z_\phi$$

where ρ' is a random group element from $\mathbb{Z}_p$. Recall that the simulator picks $\pi \in \mathbb{Z}_p$ at random prior to the game starts, and π is hidden from the adversary. Whenever the adversary makes a query that matches both selected points, the simulator picks the exponents for w and $\overline{w}$ in a correlated way such that $\overline{\gamma} = \pi\gamma$, $\overline{\gamma}_i = \pi\gamma_i$ for all $i \in \mathcal{W}(\sigma)$. It is not hard to see that $\mathsf{Game}_{4,|\overline{E}|} = \mathsf{Game}_5$.

We now prove Lemma 4.6.5, and show that a poly-time adversary cannot have more than negligible difference in its advantage in Game_4 and Game_5. Because of the hybrid argument, it suffices to show that $\mathsf{Game}_{4,d}$ is computationally indistinguishable from $\mathsf{Game}_{4,d+1}$, where $0 \leq d < |\overline{E}|$.

We prove this by supposing that a poly-time adversary $\mathcal{A}$ has more than negligible difference in its advantage against $\mathsf{Game}_{4,d}$ and $\mathsf{Game}_{4,d+1}$. Now we build a simulator $\mathcal{B}$ that leverages $\mathcal{A}$ to solve the C3DH problem.

The challenger first creates a 3-Party challenge:

$$\begin{aligned}
&(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda), \quad n \leftarrow pq, \quad g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad g_r \xleftarrow{R} \mathbb{G}_r \\
&R_1, R_2, R_3 \xleftarrow{R} \mathbb{G}_q \\
&a, b, c \xleftarrow{R} \mathbb{Z}_n \\
&\bar{Z} \leftarrow \big((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}),\ g_p,\ g_q, g_r,\ g_p^a,\ g_p^b,\quad \Gamma = g_p^{ab} \cdot R_1,\quad Y = g_p^{abc} \cdot R_2\big) \\
&Q \leftarrow g_p^c \cdot R_3
\end{aligned}$$

It then randomly decides whether to give $(\bar{Z},\ Q' = Q)$ or $(\bar{Z},\ Q' = R)$ where R is a random element in $\mathbb{G}_{pq}$.

We create the following simulation:

Init. The adversary commits two points to the simulator, X^*_0 and X^*_1. The challenger flips a random coin β internally.

Setup. Let δ denote the $d+1$-th index in $\overline{E}$.

The simulator first chooses random $(R_{u,1}, R_{h,1}), \ldots, (R_{u,\ell}, R_{h,\ell}) \in \mathbb{G}_q^3$ and random μ_1,y_1,..., μ_ℓ, $y_\ell \in \mathbb{Z}_n$.

The simulator first publishes the group description and $g_q, g_r, V = g_p R_v$, where R_v is a random element from subgroup $\mathbb{G}_r$. It picks a random $\alpha \in \mathbb{Z}_n$ and lets $A = \mathsf{e}(V, g_p)^\alpha$. It creates

$$U_\delta = g_p^{\mu_\delta} R_{u,\delta}, \quad H_\delta = g_p^{-\mu_\delta X^*_{\beta,\delta}} \Gamma^{y_\delta} R_{h,\delta}$$

Next, for all $i \neq \delta$ it creates

$$U_i = g_p^{\mu_i} R_{u,i}, \quad H_i = g_p^{-\mu_i X^*_{\beta,i}} g_p^{y_i} R_{h,i}$$

Finally, the simulator picks random $R_w, R_{\overline{w}}$ from $\mathbb{G}_q$, and random exponents $x, y, \overline{y}$ from $\mathbb{Z}_n$, and computes

$$W = g_p^x (g_p^b)^y R_w, \quad \overline{W} = (g_p^b)^{\overline{y}} R_{\overline{w}}$$

We observe that the parameters are distributed identically to the real scheme.

The simulator also sets $\pi = -y/\overline{y}$. We observe that π is information theoretically hidden from the adversary.

Query 1. Whenever the simulator receives a "reveal token" query from the adversary, it needs to compute a token of the appropriate form and return it to the adversary. The token that the adversary is requesting can be one of the following three cases: 1) non-delegated, 2) Type 1 delegated, 3) Type 2 delegated. Recall that the simulator generates Type 1 and non-delegated tokens freshly at random. Meanwhile, Section 4.6.4 provides an algorithm for generating Type 2 tokens. It suffices now to show how to generate tokens freshly at random.

Consider that the simulator has received a query from the adversary for a non-delegated token or a Type 1 delegated token σ. Recall that σ should not separate the two committed points X_0^* and X_1^*. Hence, exactly one of the following two cases must be true. Let E denote the set of indices i where the two committed points are equal, i.e., $X^*_{0,i} \neq X^*_{1,i}$, and $\overline{E} = [\ell] \backslash E$ denote the set of indices where X_0^* and X_1^* are not equal.

Case 1. $\delta \notin \mathcal{S}(\sigma) \cup \mathcal{W}(\sigma)$.

Case 2. $\delta \in \mathcal{S}(\sigma) \cup \mathcal{W}(\sigma)$. There must exist $i, j \in \mathcal{S}(\sigma)$, such that $\sigma_i \neq X_0^*$ and $\sigma_j \neq X_1^*$. In other words, the query σ does not match either of the committed identities.

Case 1. In Case 1, $\delta \notin \mathcal{S}(\sigma) \cup \mathcal{W}(\sigma)$. The simulator checks if the requested token matches both selected points. If so, the simulator picks correlated exponents for w and $\overline{w}$: $\overline{\gamma} = \pi\gamma$, and $\overline{\gamma}_i = \pi\gamma_i$ for all $i \in \mathcal{W}(\sigma)$. (Recall that the simulator sets $\pi = -y/\overline{y}$.) The simulator proceeds to generate the remaining parts of the token according to the $GenToken$ algorithm. Otherwise, if the requested token matches neither of the selected points, the simulator simply follows the $GenToken$ algorithm to generate the token. It is not hard to see that the token can be efficiently computed in this case, since the simulator knows u_i, h_i for all $i \neq \delta$, as well as other parameters needed.

Case 2. This is the more complicated case, since the simulator does not know h_δ which contains the term g_p^{ab}. Also, in this case, the token queried does not match either of the selected points. Therefore, the simulator will leverage w and $\overline{w}$ to cancel out the unknown parameters in h_δ.

We first describe how to generate the decryption key component DK. If $\delta \notin \mathcal{S}(\sigma)$, then it is trivial for the simulator to generate DK, since the unknown parameter h_δ does not appear in DK,

and the simulator knows all parameters required. If $\delta \in \mathcal{S}(\sigma)$ the simulator picks $t_\delta, \overline{\gamma}' \in \mathbb{Z}_n$ at random, and lets $\overline{\gamma}$ be the following without actually computing it.

$$\overline{\gamma} = \overline{\gamma}' - at_\delta y_\delta / \overline{y}$$

Now the simulator follows the $GenToken$ algorithm to generate remaining parts of the decryption key DK. DK can be efficiently computed, even though the simulator does not know g_p^{ab}, as all terms involving g_p^{ab} cancel out in DK. In particular, consider the term K in DK. Group the terms in K:

$$K = \left(\overline{w}^{\overline{\gamma}}(u_\delta^{\sigma_\delta} h_\delta)^{t_\delta}\right)\left(g^\alpha \prod_{j \in \mathcal{S}(\sigma), j \neq \delta} (u_j^{\sigma_j} h_j)^{t_j} Y\right)$$

The product term $\overline{w}^{\overline{\gamma}}(u_\delta^{\sigma_\delta} h_\delta)^{t_\delta}$ can be efficiently computed since all terms involving g_p^{ab} cancel out:

$$\overline{w}^{\overline{\gamma}}(u_\delta^{\sigma_\delta} h_\delta)^{t_\delta} = (g_p^b)^{\overline{y}\overline{\gamma}'} g_p^{\mu_\delta \Delta_\delta t_\delta}$$

where $\Delta_\delta = \sigma_\delta - X^*_{\beta,\delta}$. Meanwhile, the term $K_\phi = v^{\overline{\gamma}} Y_\phi$ can be efficiently computed since the simulator knows g_p^a. It is not hard to see that all remaining terms in DK can be efficiently computed.

We show how to generate the delegation components. The simulator can use exactly the same strategy to generate DL. Basically, for all $i \in \mathcal{W}(\sigma)$, the simulator picks $s_{i,\delta}, \overline{\gamma}'_i \in \mathbb{Z}_n$ at random, and lets $\overline{\gamma}_i$ be the following without actually computing it:

$$\overline{\gamma}_i = \overline{\gamma}'_i - as_{i,\delta} y_\delta / \overline{y}$$

In this way, depending on whether $\delta \in \mathcal{S}(\sigma)$ or $\delta \in \mathcal{W}(\sigma)$ the product $\overline{w}^{\overline{\gamma}_i}(u_\delta^{\sigma_\delta} h_\delta)^{s_{i,\delta}}$ or $\overline{w}^{\overline{\gamma}_i} h_\delta^{s_{i,\delta}}$ can be efficiently computed, since terms involving g_p^{ab} cancel out.

Challenge. The adversary submits two messages $\mathbf{Msg}_0$ and $\mathbf{Msg}_1$. Let $\overline{E}$ denote the set of indices i such that $X^*_{0,i} \neq X^*_{1,i}$. Let $\widetilde{E}_d$ denote the first d indices in $\overline{E}$. The simulator picks random $P \in \mathbb{G}_{pq}$, $Z_0, Z_\phi \in \mathbb{G}_q$, and $Z_i \in \mathbb{G}_q$ for all $i \in [\ell]$. The simulator creates the following ciphertext:

$$C = Q',\ \ C_0 = Q'^x P^y Z_0,\ \ C_\phi = P^{\overline{y}} Z_\phi,\ \ C_\delta = Y^{y_\delta} Z_\delta,\ \ \forall i \neq \delta \text{ and } i \notin \widetilde{E}_d:\ \ C_i = Q'^{y_i} Z_i$$

For all $i \in \widetilde{E}_d$, the simulator picks a random element in $\mathbb{G}_{pq}$ for C_i. In addition, if $\mathbf{Msg}_0 = \mathbf{Msg}_1$, the simulator computes $\widetilde{C} = \mathsf{e}(g_p, Q')^\alpha$; otherwise, the simulator replaces $\widetilde{C}$ with a random element from $\mathbb{G}_T$. Notice that if $Q' = Q$, then the above simulation is identically distributed as $\mathsf{Game}_{4,d}$. Otherwise, if $Q' = R$, the simulation is identically distributed as $\mathsf{Game}_{4,d+1}$.

Query 2. Same as phase **Query 1**.

Guess. The adversary outputs a guess β' of β. If the adversary guesses correctly, i.e., $\beta' = \beta$, the simulator guesses that $Q' = Q$ in the C3DH instance. Otherwise, the simulator guesses that $Q' = R$. It is not hard to see that any advantage of the adversary in distinguishing β translates to the simulator's advantage in solving the C3DH problem.

4.7 dHVE Full Security

We formally define the security of dHVE through the following security game between a challenger and an adversary.

- **Setup.** The challenger runs the $Setup$ algorithm, and gives the adversary the public key PK.
- **Query 1.** The adversary adaptively makes a polynomial number of "create token", "create delegated token", or "reveal token" queries. The challenger answers these queries accordingly.
- **Challenge.** The adversary outputs two pairs $(\mathbf{Msg}_0, X_0), (\mathbf{Msg}_1, X_1) \in \{0,1\}^* \times \Sigma^\ell$ subject to the following constraints:

  For any token σ revealed to the adversary in the **Query 1** stage, let $\mathcal{C}_\sigma$ denote the set of conjunctive queries corresponding to this token.

  1. For all $f \in \mathcal{C}_\sigma$, $f(X_0) = f(X_1)$.
  2. If $\exists f \in \mathcal{C}_\sigma$, $f(X_0) = f(X_1) = 1$, then $\mathbf{Msg}_0 = \mathbf{Msg}_1$.

  The challenger flips a random coin b and returns an encryption of $(\mathbf{Msg}_b, X_b)$ to the adversary.
- **Query 2.** Repeat the **Query 1** stage. All tokens revealed in this stage should satisfy the same condition as above.
- **Guess.** The adversary outputs a guess b' of b.

As before, the advantage of an adversary $\mathcal{A}$ in the above game is defined to be $\mathsf{Adv}_\mathcal{A} = |\Pr[b = b'] - 1/2|$. We say that a dHVE construction is secure if for all polynomial time adversaries, its advantage in the above game is a negligible function of λ.

4.8 Anonymous Hierarchical Identity-Based Encryption with Short Private Keys

In Section 4.1.2, we propose a new and complete security definition for delegation in these (anonymous) IBE systems. By contrast, previously, researchers have used an under-specified security game, where the adversary does not get to specify how each queried token is derived. We now show one advantage of being able to capture such nuances in our security definition, by giving an Anonymous Hierarchical Identity-Based Encryption (AHIBE) construction with shorter private keys than the original construction by Boyen and Waters [13].

To achieve this, we rely on the same technique that we use for our dHVE construction: we multiply the private keys by random group elements in the third subgroup $\mathbb{G}_r$, so that the private keys are computationally indistinguishable from being picked freshly at random.

For consistency, we build our AHIBE scheme based on composite bilinear groups and the C3DH assumption, rather than the Decisional Linear assumption adopted by the original BW construction. One can easily build the scheme using the Decisional Linear assumption as well.

In comparison, the original BW construction has $O(D^2)$ private key size and our construction has $O(D)$ private key size, where D denotes the depth of the hierarchy. Meanwhile, we preserve all other costs asymptotically, including ciphertext size, encryption cost, and decryption cost.

4.8.1 Construction

$Setup(1^\lambda, D)$: The setup algorithm takes as input a security parameter a^λ, the maximum depth $D \in \mathbb{N}$, and outputs public parameters PK and the corresponding master secret key MSK. The setup algorithm first chooses random large primes $p, q, r > m$ and creates a bilinear group $\mathbb{G}$ of composite order $n = pqr$, as specified in Section 4.3. Next, it picks a random $g, v \in \mathbb{G}_p, g_q \in \mathbb{G}_q, g_r \in G_r$, a random exponent $\alpha \in \mathbb{Z}_p$, and random elements

$$\forall n \in [0, D+1], \forall \ell \in [0, D] : \quad u_{n,\ell} \xleftarrow{R} \mathbb{G}_p$$

It keeps all the above as the master secret key MSK. The $Setup$ algorithm then chooses the following blinding factors in $\mathbb{G}_q$:

$$R_v, \quad \forall n \in [0, D+1], \forall \ell \in [0, D] : \quad R_{n,\ell} \xleftarrow{R} \mathbb{G}_q$$

The algorithm now publishes the following as the public key PK.

$$g_q, \ g_r, \ V = vR_v, \quad A = \mathsf{e}(g, v)^\alpha, \quad \forall n \in [0, D+1], \forall \ell \in [0, D] : \quad U_{n,\ell} = u_{n,\ell} R_{n,\ell}$$

$Extract(\mathsf{PK}, \mathsf{MSK}, \mathcal{I})$: The $Extract$ algorithm takes as input the public key PK, the master secret key MSK, and an ID tuple $\mathcal{I} = (I_0, I_1, \ldots, I_L) \in (\mathbb{Z}_p^\times)^{1+L}$, where $L \in [D]$, and by convention, $I_0 = 1$. The algorithm generates a private key corresponding to the identity $\mathcal{I}$.

- Pick random exponents $r_0, r_1, \ldots, r_{1+D}$ from $\mathbb{Z}_p$. Pick random blinding factors Y, Y_0, Y_1, …, Y_{1+D} from $\mathbb{G}_r$, and random $Y'_{1+L}, Y'_{2+L}, \ldots, Y'_D$ from $\mathbb{G}_r$.
- Compute the decryption key portion of the private key:

$$\mathsf{DK} = \left(K = g^\alpha \prod_{n=0}^{1+D} \prod_{\ell=0}^{L} (u_{n,\ell}^{I_n})^{r_n} \cdot Y, \quad \forall n \in [0, \ 1+D] : \quad K_n = v^{r_n} Y_n \right)$$

- Compute the following delegation components of the decryption key:

$$\mathsf{DL} = \left(\forall \ell \in [1+L, D] : \quad J_\ell = \prod_{n=0}^{1+D} u_{n,\ell}^{r_n} \cdot Y'_\ell \right)$$

$Derive(\mathsf{PK}, \mathsf{Pvk}_{\mathcal{I}|L-1}, \mathcal{I})$ The $Derive$ algorithm takes as input the public key PK, and derives a private key for $\mathcal{I} = (I_0, I_1, \ldots, I_L)$ from a parent key for $\mathcal{I}|L-1 := (I_0, I_1, \ldots, I_{L-1})$.

- First, express the parent key using the same notation as before: $\mathsf{Pvk}_{\mathcal{I}|L-1} = (\mathsf{DK}, \mathsf{DL})$, where $\mathsf{DK} = (K, K_0, K_1, \ldots, K_{1+D})$, and $\mathsf{DL} = (J_L, J_{1+L}, \ldots, J_D)$.

- Next, pick a random exponent $\tau \in \mathbb{Z}_n$, and random blinding factors $Y, Y_0, \ldots, Y_{1+D}$, and $Y'_{1+L}, \ldots, Y'_D$ from $\mathbb{G}_r$.
- Compute the decryption key portion of the child key:

$$\mathsf{DK}' = \left(K' = (K \cdot J_L^{I_L})^\tau Y, \quad \forall n \in [0,\ 1+D] :\ \ K'_n = K_n^\tau Y_n\right)$$

- Compute the delegation components of the child key:

$$\mathsf{DL}' = (\forall \ell \in [1+L, D] :\ \ J'_\ell = J_\ell^\tau Y'_\ell)$$

$Encrypt(\mathsf{PK}, \mathcal{I}, \mathbf{Msg})$ The $Encrypt$ algorithm takes a public key PK, and encrypts a message $\mathbf{Msg}$ to an identity $\mathcal{I} = (I_0, I_1, \ldots, I_L)$. The algorithm proceeds as follows:

- Pick a random exponent $s \in \mathbb{Z}_n$. Pick random blinding factors $Z, Z_0, Z_1, \ldots, Z_{1+D}$ from $\mathbb{G}_q$.
- Compute the following ciphertext:

$$\mathsf{CT} = \left(\widetilde{C} = \mathbf{Msg} A^s, \quad C = V^s Z, \qquad \forall n \in [0,\ 1+D] :\ \ C_n = (\prod_{\ell=0}^{L} U_{n,\ell}^{I_\ell})^s Z_n\right)$$

$Decrypt(\mathsf{PK}, \mathsf{Pvk}_\mathcal{I}, \mathsf{CT})$ The $Decrypt$ algorithm takes a public key PK, a private key $\mathsf{Pvk}_\mathcal{I}$, and decrypts a ciphertext CT. Using the same notation for the ciphertext and the private key as before, decrypt the message:

$$\widehat{\mathbf{Msg}} \leftarrow \frac{\widetilde{C} \cdot \prod_{n=0}^{1+D} \mathsf{e}(C_n, K_n)}{\mathsf{e}(C, K)}$$

4.8.2 Security of construction

Theorem 4.8.1 *The above-defined A-HIBE construction is internally consistent. In addition, it is IND-sID-CPA and ANON-sID-CPA secure under the cBDH and C3DH assumptions in the bilinear group* $\mathbb{G}$.

See the original BW paper [13] for detailed definitions of IND-sID-CPA and ANON-sID-CPA security.

The proof of the consistency is straightforward. Proof of security can be done in the following steps:

- As we multiply all elements of the private key with a random group element from the third subgroup $\mathbb{G}_r$, we can show that private keys generated by the $Derive$ algorithm are computationally indistinguishable from being picked freshly at random.
- Show that if private keys were really generated freshly at random rather than by calling the $Derive$ algorithm, the scheme would be IND-sID-CPA and ANON-sID-CPA secure. This part of the proof is done in a manner similar to that of the BW construction [13]. The only exception is that we now replace the Decisional Linear assumption by the C3DH assumption. However, the gist of the proof remains unchanged.

We omit the complete proof in this thesis, since it is very similar to the proof of our dHVE construction.

Chapter 5

Query Privacy in Predicate Encryption

In this chapter, we present a predicate secret-key encryption scheme that not only hides the plaintext encrypted, but also protects the privacy of the query predicates. Our construction supports inner-product queries. We begin by motivating why query privacy is an important problem.

5.1 Query Privacy in Predicate Encryption

The schemes described so far are in the public-key setting. While they guarantee the secrecy of the plaintext encrypted, they do not provide any guarantees of secrecy on the query predicate. In fact, if Alice sends Google a capability to search on her encrypted emails, Google can infer some information about the query embedded in the capability.

Leaking information about the query predicate may also be undesirable in certain applications. For example, Alice stores her encrypted documents on a remote server, and would like to perform searches on the encrypted data. Ideally, Alice would like to hide from the remote server not only her documents, but also her queries, as the queries can reveal sensitive information just like the documents. Alice could make a query for documents containing the keyword "cardiologist", which reveals her sensitive medical information. Unfortunately, in public-key predicate encryption, it is inherently impossible to guarantee the privacy of the queries (roughly in the semantic security sense). This reason is rooted in the fact that anyone can encrypt using the public key. Suppose that the server would like to learn whether a token TK corresponds to the query (DOCUMENT contains "cardiologist"), the server can take the public key, and encrypt a document containing the keyword "cardiologist". Now the server can simply apply the token TK on the resulting ciphertext to check if they match. Due to this observation, prior work on public-key predicate encryption addresses only privacy of the plaintexts (henceforth referred to as *plaintext privacy*), but not privacy of the queries (henceforth referred to as *query privacy*).

The above observation tells us *query privacy is not possible in the public-key setting*. In other words, if we would like to guarantee query privacy, we cannot let everyone have the ability to encrypt. Naturally, this raises the following question: what if we consider the secret-key setting where only the owner of the secret key can encrypt? *Is it possible to guarantee query privacy in addition to plaintext privacy in the secret-key setting?* In this chapter, we demonstrate that

it is indeed possible to achieve both query privacy and plaintext privacy in secret-key predicate encryption. Moreover, our construction supports expressive queries.

In this chapter, we present a secret-key predicate encryption scheme which guarantees both plaintext privacy and query privacy. Our construction supports inner-product queries.

Why inner-product queries? An important goal in predicate encryption is the ability to support complex, expressive queries. Researchers have made many endeavors towards this goal. The earliest schemes in the public-key setting [1, 6, 9, 13] support equality test queries such as (YEAR = 2009). Later, researchers invented schemes supporting conjunctive queries [12, 25, 37] such as (YEAR = 2009) $\wedge$ (MONTH = jan). An extension of conjunctive queries is multi-dimensional range queries [35]. Recently, Katz, Sahai and Waters [28] took another big step forward in this direction and proposed a scheme supporting inner-product queries. We point out that inner-product query is strictly more expressive than conjunctive queries. In the KSW paper [28], the authors explicitly show why inner-product queries imply conjunctions, disjunctions, CNF/DNF formulas, polynomial evaluation and exact thresholds. The KSW construction is in the public-key setting, and does not guarantee query privacy.

Naturally, a reasonable goal to aim for is a scheme whose expressiveness matches the most powerful public-key predicate encryption known to date. This is the reason why we consider inner-product queries. Our construction is the first secret-key predicate encryption scheme that guarantees query privacy and supports expressive queries.

Definitional issues. One of our contributions is to rethink the definition of query privacy. Although query privacy has previously been studied in the secret-key setting for keyword-based queries by Song et al. [39], and Curtmola et al. [19], the security definition adopted in these works are not yet satisfactory, and may be strengthened. In this thesis, we rethink how to formally define the security of Secret-Key Predicate Encryption (MRQED). Ideally, we would like to reveal the absolutely minimal information to the storage server. We capture this intuitive notion through the full security definition (see Definition 5.3.3). As the full security definition is hard to work with, we propose an alternative security definition (see Definition 5.3.4) called Single Challenge Indistinguishability (SCI). This security notion resembles the adaptive security notion adopted by previous identity-based encryption and predicate encryption schemes [3, 12, 13, 14, 15, 35]. We demonstrate in Proposition 5.3.2 that SCI security is just "as good as" full security for inner-product queries. Our construction satisfies a relaxation of SCI security. The relaxation is similar to the selective variants frequently used in prior identity-based encryption, attributed-based encryption and predicate encryption schemes [3, 12, 13, 14, 15, 35]. We emphasize that even the relaxed security model we use in our proofs is stronger than the security definitions adopted by Song et al. [39], and Curtmola et al. [19].

Proof techniques. Our proof techniques can be of independent interest. We observe that ciphertexts and tokens are *symmetric* in functionality and security requirement, and we leverage such symmetry in our proofs. More specifically, we observe that if the ciphertext and token are symmetrically formed, then by proving plaintext privacy, we obtain query privacy for free. Therefore,

one possible approach is to build a KSW-like construction, where the ciphertext and the token are symmetrically formed, or computationally indistinguishable from being symmetrically formed. In this way, we can leverage a KSW-style proof to establish plaintext privacy, and then rely on the symmetry argument to establish query privacy.

5.2 Applications of SK-PE

In the privacy-preserving Gmail example mentioned at the beginning of this thesis, it may be more appropriate to use public-key encryption, since anyone in the world should be able to use the public-key to send an encrypted email to Alice. In this case, the public-key used for encryption is known to the entire world.

On the other hand, secret-key encryption is more appropriate in other scenarios. Below, we list some potential applications of secret-key predicate encryption.

Private Google Docs In private Google Docs, Alice uses her secret key to encrypt her documents before storing them on Google Docs. Later, when Alice wishes to search these documents, she can use secret key to construct a token corresponding to her query, and send the token to Google. Using this token, Google can decide exactly which documents match Alice's search criterion, without learning any additional information. This means that Google learns nothing about the encrypted documents, and nothing about her search criterion.

Private del.icio.us `delicious.com` is a web-service allowing users to store browsing history and bookmarks, and share them with friends. Alice may not care about her privacy, if she bookmarks innocuous websites such as `movies.yahoo.com`, or `imdb.com`. However, she may care about her medical privacy, and if she wishes to bookmark the website of a hospital, she may become a little concerned about leaking this information to del.icio.us. Such privacy concerns can be addressed using secret-key predicate encryption. Alice can use her secret key to encrypt her sensitive bookmarks before storing them on del.icio.us. Later, when she wishes to search for her bookmarks, she can use her secret key to generate a token, and del.icio.us can now use this token to perform search for Alice.

5.3 Definitions: SK-PE for General Queries

Although in this thesis, we consider a specific predicate family, inner-product queries, we would like to phrase the problem of secret-key predicate encryption also in general terms. We hope that the generic definition can inspire researchers to invent secret-key predicate encryption schemes that support more powerful queries than inner products —- the version we propose in this thesis.

For simplicity, we consider the *predicate-only* version. We note that it is not hard to incorporate a payload message into the construction using techniques described in prior predicate encryption schemes [12, 28, 35].

More importantly, we rethink the security definition for SK-PE. Our security definition should capture the intuition that the remote storage server learns only Alice's access pattern, and nothing more. In particular, the storage server does not learn anything about Alice's encrypted documents, nor about what queries Alice is making.

We now give general definitions for Secret-Key Predicate Encryption (SK-PE) as well as its security. A Secret-Key Predicate Encryption (SK-PE) scheme consists of the following (possibly randomized) algorithms.

Definition 5.3.1 (Secret-key predicate encryption) *A Secret-Key Predicate Encryption (SKPE) system consists of the following (possibly randomized) algorithms.*

$Setup(1^\lambda)$: The $Setup$ algorithm takes as input a security parameter 1^λ, and outputs a secret key MSK.

$Encrypt(\mathsf{MSK}, x)$: The $Encrypt$ algorithm takes as input a secret key MSK, a plaintext $x \in \{0,1\}^\ell$; and outputs a ciphertext CT.

$GenToken(\mathsf{MSK}, f)$: The $GenToken$ algorithm takes as input a secret key MSK, and a query predicate $f : \{0,1\}^\ell \to \{0,1\}$. It outputs a token TK_f that allows one to evaluate $f(x)$ over an encryption of x. As mentioned above, we assume that the query predicate can be encoded with a bitstring of length m.

$Query(\mathsf{TK}_f, \mathsf{CT})$: The $Query$ algorithm takes as input a token TK_f for the predicate f, and a ciphertext CT which is an encryption of $x \in \{0,1\}^\ell$, the algorithm outputs $f(x)$.

5.3.1 Full Security

Public-key predicate encryption schemes guarantee the secrecy of the ciphertext; however, they do not guarantee the secrecy of the tokens. In fact, for public-key predicate encryption, it is inherently impossible to achieve ciphertext secrecy and token secrecy simultaneously. This is due to the fact that anyone is able to encrypt using the public-key. In the Gmail example, if Google would like to know whether a token corresponds to the query "TITLE = cryptography", Google can simply encrypt an email whose "TITLE = cryptography" using the public-key, and test the token against the resulting ciphertext.

In secret-key predicate encryption, it is possible to guarantee the secrecy of both the plaintext (encoded in a ciphertext) and that of the query (encoded in a token). This provides even stronger privacy guarantees in practice.

We now formally define the security for secret-key predicate encryption. As mentioned above, our definition aims to guarantee the secrecy of the plaintext, as well as the query.

To explain the intuition behind our security definition, consider a privacy-preserving remote storage application, where Alice stores her encrypted documents on a remote server, and later issues tokens to the server to search for matching documents. Our goal is to leak as little information to the storage server as possible. Under our model, Alice makes a query by submitting a token to the server, and the server learns exactly which of her encrypted documents match the query, and returns the matching documents to Alice. Therefore, in this framework, the server inevitably learns Alice's *access pattern*, a.k.a, which documents Alice retrieves with each query.

We would like to define security in the strongest sense possible: informally, *the storage server should learn only Alice's access pattern, and nothing more*. In particular, this implies that the server learns nothing about Alice's encrypted documents, or what queries she is making.

To capture the notion that the server learns only Alice's access pattern, we need to first formally define what *access pattern* means. Intuitively, the access pattern is the outcomes of q predicates on n plaintexts.

Definition 5.3.2 (Access pattern) *Let $X = (x_1, x_2, \ldots, x_n)$ denote an ordered list of n plaintexts, where $x_i \in \{0,1\}^\ell$ for $1 \leq i \leq n$. Let $F = (f_1, f_2, \ldots, f_q)$ denote an ordered list of q query predicates, where $f_i \in \{0,1\}^m$ for $1 \leq i \leq q$. The access pattern on X and F is an $q \times n$ matrix:*

$$\textsc{AccessPattern}(X, F) := \begin{bmatrix} f_1(x_1), & f_1(x_2), & \ldots, & f_1(x_n) \\ f_2(x_1), & f_2(x_2), & \ldots, & f_2(x_n) \\ \ldots, \ldots & & & \\ f_q(x_1), & f_q(x_2), & \ldots, & f_q(x_n) \end{bmatrix}$$

We now proceed to define the security for SKPE. Let $X = (x_1, x_2, \ldots, x_n)$, $X' = (x'_1, x'_2, \ldots, x'_n)$ denote two ordered lists of plaintexts. Let $F = (f_1, f_2, \ldots, f_q)$, $F' = (f'_1, f'_2, \ldots, f'_q)$ denote two ordered lists of queries predicates. Now imagine the following two worlds. In World 0, the server sees n encrypted documents $(\mathsf{Enc}(x_1), \mathsf{Enc}(x_2), \ldots, \mathsf{Enc}(x_n))$ and q tokens $\left(\mathsf{TK}_{f_1}, \mathsf{TK}_{f_2}, \ldots, \mathsf{TK}_{f_q}\right)$. In World 1, the server sees n encrypted documents $(\mathsf{Enc}(x'_1), \mathsf{Enc}(x'_2), \ldots, \mathsf{Enc}(x'_n))$ and q tokens $\left(\mathsf{TK}_{f'_1}, \mathsf{TK}_{f'_2}, \ldots, \mathsf{TK}_{f'_q}\right)$. Suppose the two worlds have the same access pattern, i.e.,

$$\textsc{AccessPattern}(X, F) = \textsc{AccessPattern}(X', F')$$

Informally, the server should not be able to distinguish between the two worlds. The security definition presented below describes a game between a challenger and an adversary, and is intended to capture this notion of indistinguishability between two these worlds. Moreover, the definition considers an adaptive adversary: an adversary who can choose what ciphertext/token queries to make depending on the previous interactions with the challenger.

Definition 5.3.3 (SKPE full security) *We say that an SKPE scheme is fully secure, if all polynomial-time adversaries have negligible advantage in the following game.*

Setup. The challenger runs the $Setup$ algorithm, and retains the secret key MSK to itself. In addition, it flips a random coin b, and keeps the bit b to itself as well. Define four ordered lists, X_0, F_0, X_1, F_1, where (X_0, F_0) will record plaintexts and predicates queried by the adversary in World 0, and (X_1, F_1) will record plaintexts and predicates queried by the adversary in World 1. Initially, all four lists are empty.

Query. The adversary adaptively makes the following types of queries. The adversary can make up to a polynomial number of these queries.

- **Ciphertext query.** The adversary specifies two plaintexts $x_0, x_1 \in \{0,1\}^\ell$ to the challenger. The challenger encrypts x_b and returns the ciphertext to the adversary. Append x_0 to the list X_0, and x_1 to the list X_1.

- **Token query.** The adversary specifies two predicates $f_0, f_1 \in \{0,1\}^m$ to the challenger. The challenger computes a token for the predicate f_{b}, and gives the resulting token to the adversary. Append f_0 to the list F_0, and f_1 to the list F_1.

All queries made in this stage should be indistinguishable by access pattern. In other words, at the end of the game, all queries made should satisfy the following condition:

$$\textsc{AccessPattern}(X_0, F_0) = \textsc{AccessPattern}(X_1, F_1)$$

Guess. The adversary outputs a guess b' of the bit b. Its advantage is defined as $\mathbf{Adv}_{\mathcal{A}} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

5.3.2 Single Challenge Indistinguishability

As the full security definition is hard to work with in our proofs, we define another security notion called Single Challenge Indistinguishability (SCI) security. For general queries, SCI security may be considered a relaxed version of the full security definition, as stated in Proposition 5.3.1. However, we show in Proposition 5.3.2 that for the specific case of inner-product queries, SCI security is as good as full security in some sense.

Definition 5.3.4 (Single Challenge Indistinguishability for general queries) *We say that an SKPE scheme (for general queries) is SCI-secure if no polynomial-time adversary has more than negligible advantage in winning the following game:*

Setup. The challenger runs the $Setup$ algorithm, and retains the secret key MSK to itself.

Query. The adversary adaptively makes the following types of queries:

- **Ciphertext query.** The adversary specifies a plaintext $x \in \{0,1\}^\ell$ to the challenger. The challenger encrypts x and returns the ciphertext to the adversary.
- **Token query.** The adversary specifies a predicate f to the challenger. The challenger computes a token for the predicate f, and gives the result to the adversary.

Challenge. The adversary requests a challenge. The adversary first specifies a bit T to the challenger.

- If $T = 0$, the challenge is a *ciphertext challenge*. The adversary then sends two plaintexts (x_0, x_1) to the challenger, satisfying the following constraint:

  Let $f_1, f_2 \ldots, f_{q_0}$ denote previously queried predicates.

  $$\forall 1 \le i \le q_0 : \quad f_i(x_0) = f_i(x_1) \tag{5.1}$$

  The challenger flips a random coin b, encrypts x_{b}, and returns the ciphertext to the adversary.
- If $T = 1$, the challenge is a *token challenge*. The adversary then sends two predicates (f_0, f_1) to the challenger, satisfying the following constraint:

Let $x_1, x_2 \ldots, x_{q_1}$ denote plaintexts that the adversary has asked the challenger to encrypt in previous ciphertext queries.

$$\forall 1 \leq i \leq q_1 : \quad f_0(x_i) = f_1(x_i) \tag{5.2}$$

The challenger flips a random coin b, computes a token $\mathsf{TK}_{f_\mathsf{b}}$ for f_b, and returns $\mathsf{TK}_{f_\mathsf{b}}$ to the adversary.

More queries. The adversary makes more queries as in the Query phase. If the adversary has previously issued a ciphertext challenge, all token queries made in this state must satisfy Equation (5.1). Otherwise, if the adversary has previously submitted a token challenge, all ciphertext queries made in this stage must satisfy Equation (5.2).

Guess. The adversary outputs a guess b' of the bit b. Its advantage is defined as $\mathbf{Adv}_{\mathcal{A}} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

5.3.3 Selective Single Challenge Indistinguishability

We now define a relaxed notion of security called Selective Single Challenge Indistinguishability, or *selective SCI* for short. Selective security has been adopted widely in the study of Identity-Based Encrytion (IBE), Anonymous Identity-Based Encryption, Attribute-based Encryption, and Public-key Predicate Encryption schemes [3, 12, 13, 14, 15, 35]. In a selective SCI game, the adversary commits to the challenge at the very beginning of the security game, and the rest of the game proceeds in the same way as the SCI security game as described in Definition 5.3.4.

Definition 5.3.5 (Selective SCI security) *We say that an SK-PE scheme is* selectively *SCI-secure if no polynomial-time adversary has more than negligible advantage in winning the following game:*

Init. The adversary submits a challenge to the challenger. Like before, the challenge is composed of a bit T indicating whether this is a ciphertext challenge or a token challenge; followed by two plaintexts (x_0, x_1) (in the case of a ciphertext challenge), or two query predicates (f_0, f_1) (in the case of a token challenge).

Setup. The challenger runs the $Setup$ algorithm, and retains the secret key MSK to itself.

Query. The adversary adaptively makes either ciphertext queries or token queries, and the challenger responds to the queries accordingly. If the adversary has previously issued a ciphertext challenge, all token queries made in this stage must satisfy Equation (5.1). Otherwise, if the adversary has previously submitted a token challenge, all ciphertext queries made in this stage must satisfy Equation (5.2).

Challenge. The challenger flips a random coin b, and returns either an encryption of x_b, or a token for the predicate f_b depending on the type of challenge specified by the adversary in the Init stage.

More queries. The adversary makes more queries as in the Query phase. If the adversary has previously issued a ciphertext challenge, all token queries made in this state must satisfy Equation (5.1). Otherwise, if the adversary has previously submitted a token challenge, all ciphertext queries made in this stage must satisfy Equation (5.2).

Guess. The adversary outputs a guess b' of the bit b; and its advantage is defined as $\mathbf{Adv}_{\mathcal{A}} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

5.3.4 Relationship Between Security Definitions

For general queries, it is not hard to see that full security implies SCI security. Therefore, we can consider SCI security as a relaxed version of the full security definition. Interestingly, we are able to show that for the special case of inner-product queries, *given a scheme satisfying SCI security for vectors of length $2n$, we can construct a fully-secure scheme for vectors of length n.* We refer the readers to Proposition 5.3.2 for a more formal statement and proof of this observation. Proposition 5.3.2 tells us that to construct a fully-secure SK-PE scheme on inner-product queries, it suffices to construct a scheme satisfying SCI security.

Proposition 5.3.1 *If an SK-PE scheme (for general queries) is fully-secure, it must be SCI-secure.*

Proof: (*sketch.*) Notice that an SCI adversary is a special case of an adversary in the full-security game. In the full-security game, suppose that in all but one query, the adversary submits two equal plaintexts (or queries), i.e., $x_0 = x_1$ (or $\delta_0 = \delta_1$), then the adversary is in fact an SCI adversary. ■

In the special case of inner-product queries, SCI security is "as good as" full security, and the following proposition explains why.

Proposition 5.3.2 *Let* SCHEME_{2n} *denote an SCI-secure SK-PE scheme supporting inner-product queries, where both plaintext and query vectors have length $2n$ (i.e., plaintext and query vectors are picked from Σ^{2n}). Given* SCHEME_{2n}*, it is possible to construct a fully-secure SK-PE scheme supporting inner-product queries, where both the plaintext and query vectors have length n. We refer to the latter scheme as* SCHEME_n.

While the detailed proof of the above proposition is provided in Section 5.8, we explain the intuition here. In the full security game, the challenger constructs ciphertexts and tokens for different vectors in World 0 and World 1. Suppose that the challenger encrypts vectors $X = (\vec{x}_1, \ldots, \vec{x}_c)$ and constructs tokens for vectors $V = (\vec{v}_1, \ldots, \vec{v}_t)$ in World 0; the challenger encrypts vectors $Y = (\vec{y}_1, \ldots, \vec{y}_c)$ and constructs tokens for vectors $W = (\vec{w}_1, \ldots, \vec{w}_t)$ in World 1. It is required that the access pattern remains the same between these two worlds, that is, $\text{ACCESSPATTERN}(X, V) = \text{ACCESSPATTERN}(Y, W)$.

If we could define a sequence of hybrid games in between World 0 and World 1, such that only one component (one ciphertext or one token) is changed between any two consecutive games, then we would be able to prove full security using SCI security plus a hybrid argument. By the definition of SCI security, a computationally-bounded adversary is unable to distinguish between two games where only one component differs (as long as these two worlds have the same access pattern). Unfortunately, we cannot naively change any component alone in World 0, since doing so might result in a different access pattern. For example, suppose the challenger changed from encrypting $\vec{x}_c$ to encrypting $\vec{y}_c$ in World 0, then this might cause the access pattern to change as well. To solve this problem, we propose to encrypt the vector $\vec{x}$ twice. More specifically, to encrypt $\vec{x}$, we encrypt the length $2n$ vector

$$\vec{x}||\vec{x} := (x_1, x_2, \ldots, x_n, x_1, x_2, \ldots, x_n)$$

instead, using the SCI-secure construction SCHEME_{2n}. Similarly, to construct a token for the vector $\vec{v}$, we construct a token for the length $2n$ vector $\vec{v}||\vec{v}$ instead. In this way, we construct SCHEME_n (for vectors of length n) from an SCI-secure SCHEME_{2n} (for vectors of length $2n$). As Section 5.8 demonstrates, this allows us to define a sequence of hybrid games where only one component is changed between any two consecutive games. Meanwhile, the access pattern is preserved across all games. See Section 5.8 for the detailed proof of this proposition.

A note on selective SCI security. Our construction is proven secure under the selective SCI model. One way to interpret the strength of selective SCI security is as follows. We have explained that selective SCI security is a relaxation of SCI security. Meanwhile, as Proposition 5.3.2 points out, SCI security is "as good as" full security for inner-product queries. Therefore, we can informally think of selective SCI security as a relaxation of full security for inner-product queries. The selective security model has frequently been adopted in prior IBE, ABE and predicate encryption schemes. We emphasize that even this relaxed security model is better than the definitions previously adopted. In particular, we show in Section 5.9 that given an SK-PE scheme for vectors of length $2n$ satisfying selective SCI security, one can construct an SK-PE scheme for vectors of length n whose security is strictly stronger than the definition previously adopted by Curtmola et al. [19]. Curtmola et al. studied SK-PE for keyword-based queries, and proposed one possible formalization of query privacy.

5.4 Background on Pairings and Complexity Assumptions

5.4.1 Bilinear groups of composite order

We review some background on bilinear maps and groups, especially groups of *composite order*, which were first introduced by Boneh, Goh and Nissim [10].

Let GG denote a *group generator* algorithm which takes as input a security parameter $\lambda \in \mathbb{Z}^{>0}$, a number $k \in \mathbb{Z}^{>0}$, and outputs a tuple $(p_1, p_2, \ldots p_k, \mathbb{G}, \mathbb{G}_T, \mathsf{e})$ where $p_1, p_2, \ldots p_k$ are k distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are two cyclic groups of order $n = \prod_{i=1}^{k} p_i$. The function $\mathsf{e} : \mathbb{G}^2 \rightarrow \mathbb{G}_T$ satisfies the following properties:

- (Bilinear) $\forall u, v \in \mathbb{G}$, $\forall a, b \in \mathbb{Z}$, $\mathsf{e}(u^a, v^b) = \mathsf{e}(u, v)^{ab}$.
- (Non-degenerate) $\exists g \in \mathbb{G}$ such that $\mathsf{e}(g, g)$ has order n in $\mathbb{G}_T$.

We assume that group operations in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the bilinear map e can be computed in time polynomial in λ. We use the notation $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}, \ldots, \mathbb{G}_{p_k}$ to denote the respective subgroups of order $p_1, \ldots, p_k$ of $\mathbb{G}$. We use $\mathbb{G}_{p_1 p_2}, \mathbb{G}_{p_2 p_5 p_k}$ to denote the subgroups of order $p_1 p_2$ and $p_2 p_5 p_k$ respectively. For example, $\mathbb{G}_{p_1 p_2} = \mathbb{G}_{p_1} \times \mathbb{G}_{p_2}$.

5.4.2 Our assumptions

The predicate-only version of our construction relies on three assumptions, Assumption 1 in the KSW paper [28], the generalized 3-party Diffie-Hellman assumption (C3DH), and the Decisional

Linear (DL) assumption. All of these assumptions involves at most 3 subgroups simultaneously. In particular, Assumption 1 involves 3 subgroups, C3DH involves 2 subgroups, and DL involves 1 subgroup simultaneously. We assume that these assumptions still hold when the relevant subgroup(s) fall within a larger group whose order is the product of 4 distinct primes, $N = pqr\widehat{r}$. Moreover, the naming of the subgroups is not significant in our assumptions, that is, the same assumptions still hold after renaming the subgroups.

Assumption 1 of KSW [28]. Our scheme is built on top of the KSW construction [28]. As a result, we inherit their complexity assumptions as well. In particular, the predicate-only version relies on Assumption 1 of the KSW construction.

We assume that this assumption holds when $\mathbb{G}_p \times \mathbb{G}_q \times \mathbb{G}_r$ belongs to a larger group of order $N = pqr\widehat{r}$; and below, we restate it in the context of the larger group.

Assumption 1 posits that any polynomial-time adversary has a negligible advantage in the following experiment: Let $N = pqr\widehat{r}$, let $g_p, g_q, g_r, \widehat{g}_r$ be random generators of $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_{\widehat{r}}$ respectively. Pick the following numbers at random: $Q_1, Q_2, Q_3 \in \mathbb{G}_q$, $R_1, R_2, R_3 \in \mathbb{G}_r$, $a, b, s \in \mathbb{Z}_p$, and a random bit b. Give the adversary the description of the bilinear group $(N, \mathbb{G}, \mathbb{G}_T, \mathsf{e})$, and the following set of values:

$$S = \left\{ g_p,\ \ g_r,\ \ \widehat{g}_r,\ \ g_q R_1,\ \ g_p^b,\ \ g_p^{b^2},\ \ g_p^a g_q,\ \ g_p^{ab} Q_1,\ \ g_p^s,\ \ g_p^{bs} Q_2 R_2 \right\} \tag{5.3}$$

In addition, if $\mathsf{b} = 0$, the adversary is given the value $T = g_p^{b^2 s} R_3$; otherwise, if $\mathsf{b} = 1$, the adversary is given the value $T = g_p^{b^2 s} Q_3 R_3$. The adversary outputs a guess b' of the bit b, and its advantage is defined as

$$\mathbf{Adv}_{\mathcal{A}} = \left| \Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2} \right|$$

Assumption 1 states that no polynomial-time adversary can win this game with more than negligible advantage. Note that this assumption implies the hardness of factoring N.

Generalized 3-party Diffie-Hellman assumption (C3DH). We also rely on the composite 3-party Diffie-Hellman assumption first introduced by Boneh and Waters [12]. We restate the assumption in the context of a bilinear group whose order is the product of four distinct primes $N = pqr\widehat{r}$.

Let $g_p, g_q, g_r, \widehat{g}_r$ denote random generators from the subgroups $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_{\widehat{r}}$ respectively. Let R_1, R_2, R_3 denote random elements from the subgroup $\mathbb{G}_r$, let a, b, c denote random exponents from $\mathbb{Z}_N$. Now a challenger gives an adversary the following values:

$$\left(g_p,\ g_q,\ \ g_r,\ \widehat{g}_r,\ \ \ g_p^a,\ \ g_p^b,\ \ g_p^{ab} \cdot R_1,\ \ \ g_p^{abc} \cdot R_2 \right)$$

The challenger also flips a random coin b, and depending on the value of b, the challenger gives the adversary either the value $g_p^c \cdot R_3$ or a random element from the subgroup $\mathbb{G}_{pr}$. The adversary's task is to output a guess b' of the bit b, and its advantage is defined as $\mathbf{Adv}_{\mathcal{A}} = \left| \Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2} \right|$.

The C3DH assumption posits that for any polynomial time algorithm $\mathcal{A}$, its advantage in the C3DH experiment is a negligible function. Note that this assumption implies the hardness of factoring N.

Decisional Linear assumption (DL). We also rely on the Decisional Linear assumption first used by Boneh, Boyen and Shacham for group signatures [5]. Below we restate the assumption in the context of a larger group whose order is the product of four distinct primes $N = pqr\widehat{r}$.

Let $\mathbb{G}_p$ denote the subgroup of order p in a bilinear group G of order $N = pqr\widehat{r}$. The adversary is given

$$(g_p, g_q, g_r, \widehat{g}_r, g_p^{z_1}, g_p^{z_2}, g_p^{z_1 z_3}, g_p^{z_2 z_4})$$

where z_1, z_2, z_3, z_4 are picked at random from $\mathbb{Z}_p$, and $g_p, g_q, g_r, \widehat{g}_r$ are random generators of the subgroups $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r$ and $\mathbb{G}_{\widehat{r}}$ respectively. In addition, the adversary is given either $Z = g_p^{z_3+z_4}$, or a random element from $\mathbb{G}_p$. The adversary's task is to distinguish between these two cases.

The Decisional Linear assumption posits that no polynomial-time adversary has more than negligible advantage in the above experiment.

5.5 Construction

In this section, we propose an SK-PE construction for inner-product queries. A plaintext $\vec{x}$ is a vector drawn from $\mathbb{Z}_N^n$. A query predicate represented as $\vec{v}$ is also drawn from $\mathbb{Z}_N^n$. A predicate vector $\vec{v}$ specifies the following predicate function:

$$f_{\vec{v}} = \begin{cases} 0 & \text{if } \langle \vec{x}, \vec{v} \rangle = 0 \\ 1 & \text{otherwise} \end{cases}$$

5.5.1 Intuition

Recall that our goal is to construct a scheme supporting inner-product queries in the secret-key setting. Furthermore, we aim to achieve both plaintext and query privacy. As the KSW construction [28] already provides a solution for inner-product queries in the public-key setting, our first attempt is to directly use the KSW construction. We can conveniently convert the KSW construction to the secret-key setting, simply by withholding the public-key. This approach immediately ensures plaintext privacy as proven in the KSW paper. Unfortunately, the KSW construction does not provide any guarantee about query privacy; in fact, as we point out in Section 5.1, query privacy is not possible in the public-key setting. Therefore, it seems that our biggest challenge is how to achieve query privacy. We now explain how we can rely on the symmetry observation to address this challenge.

Observe that the ciphertext and the token are completely symmetric. In terms of functionality, both the plaintext and query are vectors of length n; meanwhile, the inner-product equation is commutative. In terms of security definitions, the ciphertext and the token are symmetric as well. The ciphertext needs to hide the plaintext vector, while the token needs to hide the query vector. One way to interpret the symmetry is to think of the ciphertext as an encryption of the plaintext vector, and think of the token as an encryption of the query vector. In fact, under the definitions given in Section 5.3, we can safely reverse the role of a ciphertext and a token. In other words, we can have tokens serve as ciphertexts, and ciphertexts serve as tokens.

This symmetry observation gives rise to the following idea: what if we construct a scheme where the ciphertexts and tokens are symmetrically formed? This can make life much simpler for us, since if we are able to prove plaintext privacy, we will obtain query privacy for free. Due to the symmetry in formation, the same argument we use to prove plaintext privacy can be used to prove query privacy as well. In our construction, the ciphertext and the token are not exactly symmetric by formation, however, we prove that ciphertext and tokens are, in fact, computationally indistinguishable from being symmetric. In other words, a computationally-bounded adversary is unable to distinguish our scheme from another scheme (called SCHEMESYM in the proof) where the ciphertext and the token are symmetric by distribution. We henceforth refer to this as *computational symmetry*.

We now present our main construction, and then, in Section 5.5.3, we explain at the algebraic level: (1) why our construction has computational symmetry, and (2) how to understand the differences between our construction and KSW, and why these differences are important to ensure query privacy.

5.5.2 Detailed construction

We now present our main construction.

$Setup(1^\lambda)$: The setup algorithm first chooses random large primes $p, q, r, \widehat{r}$, and creates a bilinear group of composite order $N = pqr\widehat{r}$. Next it picks generators $g_p,\ g_q,\ g_r,\ \widehat{g}_r$ from subgroups $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_{\widehat{r}}$ respectively. It then picks $h_{1,i}, h_{2,i}, \bar{h}_{1,i}, \bar{h}_{2,i}$ from $\mathbb{G}_p$, for all $1 \le i \le n$.

The secret key is set to the following:

$$\mathsf{Pvk} = \left(g_p,\ g_q,\ g_r,\ \widehat{g}_r, \quad \left\{h_{1,i}, h_{2,i}, \bar{h}_{1,i}, \bar{h}_{2,i}\right\}_{i=1}^{n}\right)$$

$Encrypt(\mathsf{MSK}, \vec{x})$: Let $\vec{x} = (x_1, x_2, \ldots, x_n) \in (\mathbb{Z}_N)^n$. The encryption algorithm first picks random exponents s, t, α, β from $\mathbb{Z}_N$. Then, it chooses random hiding factors $\widehat{R}_0, \widehat{R}_\emptyset$ from the subgroup $\mathbb{G}_{\widehat{r}}$; and random $\{R_{1,i}, R_{2,i}\}_{i=1}^n$ from $\mathbb{G}_r$.

Next, the encryption algorithm computes the following ciphertext:

$$\mathsf{CT} = \left(\begin{array}{l} C_0 = \widehat{R}_0 \cdot g_p^s, \quad C_\emptyset = \widehat{R}_\emptyset \cdot g_p^t \\ \left\{C_{1,i} = h_{1,i}^s \bar{h}_{1,i}^t g_q^{\alpha x_i} R_{1,i},\ \ C_{2,i} = h_{2,i}^s \bar{h}_{2,i}^t g_q^{\beta x_i} R_{2,i}\right\}_{i=1}^n \end{array} \right)$$

$GenToken(\mathsf{MSK}, \vec{v})$: Let $\vec{v} = (v_1, v_2, \ldots, v_n) \in (\mathbb{Z}_N)^n$. The $GenToken$ algorithm picks random exponents $f_1, f_2, \{r_{1,i}, r_{2,i}\}_{i=1}^n$ from $\mathbb{Z}_N$. Then, it chooses random hiding factors $R_0, R_\emptyset$ from the subgroup $\mathbb{G}_r$; and random $\{\widehat{R}_{1,i}, \widehat{R}_{2,i}\}_{i=1}^n$ from $\mathbb{G}_{\widehat{r}}$.

Next, the $GenToken$ algorithm computes the following token:

$$\mathsf{TK} = \left(\begin{array}{l} K_0 = R_0 \cdot \prod_{i=1}^n h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \quad K_\emptyset = R_\emptyset \cdot \prod_{i=1}^n \bar{h}_{1,i}^{-r_{1,i}} \bar{h}_{2,i}^{-r_{2,i}} \\ \left\{K_{1,i} = g_p^{r_{1,i}} g_q^{f_1 v_i} \widehat{R}_{1,i},\ \ K_{2,i} = g_p^{r_{2,i}} g_q^{f_2 v_i} \widehat{R}_{2,i}\right\}_{i=1}^n \end{array} \right)$$

$Query(\mathsf{TK}_{\vec{v}}, \mathsf{CT}_{\vec{x}})$: The $Query$ algorithm behaves in a way similar to the KSW [28] construction. It computes

$$\mathsf{e}(C_0, K_0)\mathsf{e}(C_\emptyset, K_\emptyset) \cdot \prod_{i=1}^{n} \mathsf{e}(C_{1,i}, K_{1,i})\mathsf{e}(C_{2,i}, K_{2,i}) \stackrel{?}{=} 1 \tag{5.4}$$

and outputs 0 iff the above is equal to 1, indicating that $\langle \vec{x}, \vec{v} \rangle = 0 \mod N$. (The case that $\langle \vec{x}, \vec{v} \rangle = 0 \mod q$, but $\langle \vec{x}, \vec{v} \rangle \neq 0 \mod N$ happens with negligible probability as explained in Section 5.6.)

5.5.3 How to understand our construction

Computational symmetry. As mentioned in Section 5.5.1, the ciphertexts and tokens in our construction are computationally symmetric. To understand our computational symmetry idea, it helps to observe the following facts when inspecting our construction. (1) The $\mathbb{G}_q$ subgroup is completely symmetric. In the ciphertext, the $\mathbb{G}_q$ subgroup encodes the plaintext vector, while in the token, the $\mathbb{G}_q$ subgroup encodes the query vector. (2) The $\mathbb{G}_r$ subgroup and the $\mathbb{G}_{\widehat{r}}$ subgroups behave as mirrors of each other. Whenever an element from $\mathbb{G}_r$ appears in the ciphertext, an element from $\mathbb{G}_{\widehat{r}}$ appears in the corresponding term in the token, and vice versa. (3) The $\mathbb{G}_p$ subgroup is not completely symmetric in the ciphertext and the token, however, we later prove that the $\mathbb{G}_p$ subgroup appears to be symmetric to a computationally-bounded adversary.

Comparison with the KSW construction. Since KSW already proved plaintext privacy for inner-product queries in the public-key setting, we tried to build a construction resembling KSW, in hope of reusing their proof (or proof techniques) on plaintext privacy. To aid the understanding of our construction, we provide a review of the KSW construction in Section 5.10.

What is more interesting to the reader might be the differences between our construction and the KSW construction. In fact, a good way to understand our scheme is to compare it with the KSW construction. We now explain the important differences from the KSW construction that are crucial in achieving query privacy.

- *The $\mathbb{G}_{\widehat{r}}$ subgroup.* The KSW construction relies on 3 subgroups, $\mathbb{G}_p$, $\mathbb{G}_q$ and $\mathbb{G}_r$. We introduce an additional subgroup $\mathbb{G}_{\widehat{r}}$, whose order $\widehat{r}$ is a large prime distinct from p, q and r. The most important functionality of the subgroup $\mathbb{G}_{\widehat{r}}$ is to serve as random hiding factors for most terms in the token. Intuitively, these random hiding factors can hide the query vector encoded in the token, thereby achieving query privacy. The behavior of the $\mathbb{G}_{\widehat{r}}$ subgroup "mirrors" that of the $\mathbb{G}_r$ subgroup. Consequently, the $\mathbb{G}_{\widehat{r}}$ also helps to introduce symmetry into our construction.
- *The $\mathbb{G}_p$ subgroup.* In the KSW construction, all terms in the ciphertext have the same exponent s in the $\mathbb{G}_p$ subgroup. By contrast, we introduce an extra degree of randomness represented by the exponent t. Terms in the ciphertext now rely on two degrees of randomness, namely, s and t, in the $\mathbb{G}_p$ subgroup. Informally, this change is due to the observation that the $\mathbb{G}_p$ subgroup is asymmetric in the ciphertext and the token by formation. Moreover, having only one degree of randomness (like in the KSW construction) is insufficient to ensure "computational symmetry"

in the $\mathbb{G}_p$ subgroup. However, if we increase the degree of randomness to two, then we can show that the $\mathbb{G}_p$ subgroup is computationally symmetric in the ciphertext and the token.

To understand why this is the case, recall that Diffie-Hellman is easy in bilinear groups. Another interpretation of this statement is that if we pick a vector $g_p^{\alpha_1}, g_p^{\alpha_2}, \ldots, g_p^{\alpha_k}$, it is easy to decide whether the exponent vector $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ are picked independently at random, or picked from a prescribed one-dimensional subspace. On the other hand, an informal interpretation of the Decisional Linear assumption tells us that it is computationally hard to decide whether the exponent vector $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ are picked independently at random, or picked randomly from a prescribed 2-dimensional subspace. The reason for introducing the extra randomness t in the ciphertext is exactly to ensure that the exponents in the $\mathbb{G}_p$ subgroup are picked from a 2-dimensional subspace, rather than a 1-dimensional subspace. This is why our construction has computational symmetry in the $\mathbb{G}_p$ subgroup.

- *The Q element in K.* The careful reader may have noticed that in the original KSW construction, the first term in the token K has a $Q \in \mathbb{G}_q$ element. This Q element, however, has disappeared from the K_0 and $K_\emptyset$ terms in our construction. (Notice that the analog of KSW's K term is K_0 and $K_\emptyset$ in our construction. The extra $K_\emptyset$ term results from introducing the extra randomness t into the ciphertext.) The Q term seems indispensable in the KSW construction if one carefully examines their proof. Consequently, the fact that we can remove the Q term may seem counter-intuitive at first. However, we are able to show that whether K_0 and $K_\emptyset$ terms contain an element from the $\mathbb{G}_q$ subgroup is computationally indistinguishable to a polynomial-time adversary. It turns out that the ability to remove the Q term is a side benefit from the introduction of the $\mathbb{G}_{\widehat{r}}$ subgroup into the tokens. As a result, our proof does *not* indicate that it is safe to remove the Q term from the KSW construction as well.

  Moreover, the ability to remove the Q term helps to introduce symmetry to our construction. Clearly, in the KSW construction, the Q term is one conspicuous place where the ciphertext and the token do not mirror each other.

5.5.4 Security and proof overview

Theorem 5.5.1 *Under the generalized Assumption 1 of the KSW construction [28], the generalized C3DH assumption, and the Decisional Linear assumption, our main construction (Section 5.5) is selectively SCI-secure against polynomial-time adversaries.*

We now give an overview of our security proof. Apart from this section, Section 5.5.1 also sheds light on the intuition behind our construction and proofs.

In the proof, we present two variants of the main construction, SCHEMESYM and SCHEMEQ. We refer to our main construction as SCHEMEREAL. We prove that SCHEMEREAL is computationally indistinguishable from both SCHEMESYM and SCHEMEQ. We now explain the motivation for having the two variants SCHEMESYM and SCHEMEQ.

SCHEMESYM. Recall that we plan to use computational symmetry in our construction proof. In particular, if ciphertexts and tokens are symmetrically formed in our construction, we will only

need to prove plaintext privacy, and we get query privacy for free. The same argument also applies if the ciphertexts and tokens are not symmetric by distribution, but *computationally symmetric*. SCHEMESYM is exactly the variant where ciphertexts and tokens are symmetrically formed by distribution. And as our construction is computationally indistinguishable from SCHEMESYM, it means that in our construction, ciphertexts and tokens are computationally symmetric.

SCHEMEQ. Now we have proven the computational symmetry between the formation of the ciphertexts and tokens, it remains to prove plaintext privacy. To this end, we would like to reuse KSW's proof on plaintext privacy. If our construction was close enough to the KSW construction, we might be able to reuse their proof as a blackbox, without having to re-invent the wheel. We give a review of the KSW construction in Section 5.10.

A big difference between our construction SCHEMEREAL and the KSW construction is that to ensure the symmetry property, we have removed the $\mathbb{G}_q$ subgroup from the K_0 and $K_\emptyset$ terms (which correspond to the K term in the KSW construction) in the token. The purpose of SCHEMEQ is exactly to restore the missing elements $Q_0, Q_\emptyset \xleftarrow{R} \mathbb{G}_q$ to the K_0 and $K_\emptyset$ terms. By restoring these elements, we obtain a scheme that bears sufficient resemblance to KSW, such that we can reuse KSW's proof as a blackbox. Specifically, we show that if an adversary can break the plaintext privacy of SCHEMEQ, we can leverage that adversary to break the plaintext privacy of the KSW construction as well. In addition, as our main construction SCHEMEREAL is computationally indistinguishable from SCHEMEQ, the plaintext privacy of SCHEMEQ immediately carries over to SCHEMEREAL.

Another perspective. As mentioned above, we have three variants in the proof, our main construction SCHEMEREAL, a symmetric construction SCHEMESYM, and a construction with the $Q_0, Q_\emptyset$ terms restored called SCHEMEQ. In fact, we show that all three variants are computationally indistinguishable from each other. This means that the properties we prove on one variant automatically carry over to the other two variants. In fact, all three variants have symmetric or computationally symmetric ciphertexts and tokens; and all three variants have plaintext privacy. As a result, all three variants have query privacy as well. This also suggests that any of these three schemes can be our main construction. The reason why we chose SCHEMEREAL to be our main construction is merely due to the fact that SCHEMEREAL is easier to express and slightly faster to compute than the other two variants.

An alternative way to interpret our proof is as follows. Suppose that we used SCHEMESYM as our main construction instead. Our goal is to prove that SCHEMESYM has both plaintext and query privacy. As ciphertexts and tokens are symmetric by distribution in SCHEMESYM, it suffices to prove plaintext privacy of SCHEMESYM. And to prove the plaintext privacy of SCHEMESYM, we show that SCHEMESYM is computationally indistinguishable from SCHEMEQ, and that SCHEMEQ has plaintext privacy. However, to show that SCHEMESYM is computationally indistinguishable from SCHEMEQ, we need to introduce an intermediate step: first, show that SCHEMESYM is computationally indistinguishable from SCHEMEREAL; then, show that SCHEMEREAL is computationally indistinguishable from SCHEMEQ.

We defer the detailed proof of Theorem 5.5.1 to Section 5.7.

5.6 Correctness

The correctness of the above construction relies on the following facts, which tells us that no cross-subgroup interaction happens when we perform a pairing operation on two group elements. Although the following facts are stated using the notations $\mathbb{G}_p$ and $\mathbb{G}_q$, they also apply to general composite-order bilinear groups.

Fact 5.6.1 *Let $a_p \in \mathbb{G}_p, b_q \in \mathbb{G}_q$ denote two elements from distinct subgroups. Then $\mathsf{e}(a_p, b_q) = 1$.*

From Fact 5.6.1 and the bilinear property of the pairing function e, we can derive the following fact.

Fact 5.6.2 *Let $\mathbb{G}_{pq} = \mathbb{G}_p \times \mathbb{G}_q$, $a, b \in \mathbb{G}_{pq}$. a and b can be rewritten (uniquely) as $a = a_p a_q$, $b = b_p b_q$, where $a_p, b_q \in \mathbb{G}_p$, and $a_q, b_q \in \mathbb{G}_q$. Furthermore,*

$$\mathsf{e}(a, b) = \mathsf{e}(a_p, b_p)\mathsf{e}(a_q, b_q)$$

In plain English, this means that when we perform a pairing operation on a and b, there is no cross-subgroup interaction. It is equivalent to performing a pairing inside each subgroup and multiplying the results together.

Now we can check the correctness of the $Query$ algorithm. It is not hard to see that in Equation (5.4), operations in the subgroups $\mathbb{G}_p, \mathbb{G}_r, \mathbb{G}_{\hat{r}}$ all result in $1 \in \mathbb{G}_T$. Therefore, we only need to focus on the subgroup $\mathbb{G}_q$; and the outcome of Equation (5.4) is:

$$\mathsf{e}(g_q, g_q)^{(\alpha f_1 + \beta f_2)\langle \vec{x}, \vec{v} \rangle}$$

Therefore, if $\langle \vec{x}, \vec{v} \rangle = 0 \mod N$, then the above evaluates to 1. Otherwise, if $\langle \vec{x}, \vec{v} \rangle \neq 0 \mod N$, there are two cases: (a) $\langle \vec{x}, \vec{v} \rangle = 0 \mod q$. This case reveals a non-trivial factor of N, and therefore, happens with negligible probability. (b) $\langle \vec{x}, \vec{v} \rangle \neq 0 \mod q$. In this case, except with negligible probability, $\alpha f_1 + \beta f_2 \neq 0 \mod q$, and the output of Equation (5.4) is not equal to $1 \in \mathbb{G}_T$.

5.7 Security Proof

5.7.1 Terminology used in the proof

We now prove the selective SCI security (Definition 5.3.5) of our construction. To do this, it suffices to prove *plaintext privacy* and *query privacy* separately.

Definition 5.7.1 (Selective plaintext privacy) *An adversary plays the security game in Definition 5.3.5 with a challenger. However, the adversary submits ciphertext challenges only. An SK-PE scheme has selective plaintext privacy, iff no polynomial-time adversary can win the security game with more than negligible advantage.*

Definition 5.7.2 (Selective query privacy) *Selective query privacy is similarly defined as selective plaintext privacy, except that now the adversary can only submit token challenges in the security game.*

It is not hard to see that to prove selective SCI security, it suffices to prove that the scheme has both selective plaintext privacy and selective query privacy, as stated in the following lemma.

Lemma 5.7.3 *An SK-PE scheme that has both selective plaintext privacy and selective query privacy is selective SCI-secure.*

In the proof, we often need modifications to our main construction, and show that the resulting encryption scheme is *computationally indistinguishable* from the original construction. To prove that the original construction has a certain security property, it suffices to prove that the new scheme has that security property. The following definition formally states what it means for two encryption schemes to be computationally indistinguishable.

Definition 5.7.4 (Indistinguishability of encryption schemes) *We say that two SK-PE encryption schemes* SCHEMEA *and* SCHEMEB *are computationally indistinguishable from each other, if no polynomial-time adversary has more than negligible advantage in winning the following distinguishing game:*

- **Setup.** The challenger flips a random coin b. If $b = 1$, SCHEMEA is chosen; otherwise, SCHEMEB is chosen. Now the challenger runs the setup algorithm of the chosen scheme, and retains the secret key MSK to itself.
- **Queries.** The adversary adaptively makes ciphertext queries and token queries. In other words, the adversary can request that the challenger reveal an encryption of a plaintext $\vec{x}$ of its choice or request that the challenger reveal a token for $\vec{v}$ of its choice. The challenger computes the requested ciphertext (token) according to SCHEMEA or SCHEMEB depending on which one has been chosen.
- **Guess.** At the end of the distinguishing game, the adversary guesses which encryption scheme has been chosen, i.e., it outputs a guess b' of the bit b chosen by the challenger. The adversary's advantage is defined as $\mathbf{Adv}_{\mathcal{A}} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

The notion of computational indistinguishability between two encryption schemes will be useful throughout our proofs, as to prove plaintext privacy (or query privacy) of SCHEME1, it suffices to prove plaintext privacy (or query privacy) of its counterpart SCHEME2 which is computationally indistinguishable from SCHEME1. This is formally stated in the proposition below.

Proposition 5.7.1 *Let* SCHEME1 *and* SCHEME2 *denote two SK-PE schemes that are computationally indistinguishable from each other. If* SCHEME1 *has plaintext privacy (or query privacy), then* SCHEME2 *must have plaintext privacy (or query privacy) as well.*

Proof: (*sketch*.) Suppose for the purpose of a contradition that SCHEME1 has plaintext privacy, but SCHEME2 does not have plaintext privacy. This means that there exists a polynomial-time adversary $\mathcal{A}$ who can win the plaintext privacy game (of SCHEME2) with non-negligible probability ϵ. We can now leverage this adversary $\mathcal{A}$ to distinguish SCHEME1 and SCHEME2. We build a simulator $\mathcal{B}$. When given an encryption scheme SCHEME, $\mathcal{B}$ can decide whether SCHEME is SCHEME1 or SCHEME2 with probability at least $\epsilon/2$. The simulator's strategy is to play the plaintext privacy game with $\mathcal{A}$, and if $\mathcal{A}$ wins the plaintext privacy game, our simulator outputs SCHEME2; otherwise, it outputs SCHEME1. This contradicts with the assumption that SCHEME1 and SCHEME2 are computationally indistinguishable. ■

5.7.2 Proof overview

The proof consists of two parts.

1. We first show in Section 5.7.3 that our main construction (henceforth referred to as SCHEMEREAL) guarantees selective plaintext privacy. This part of the proof is done in two steps. First, we show that SCHEMEREAL is computationally indistinguishable from a variant scheme called SCHEMEQ. Second, we show that SCHEMEQ has selective plaintext privacy. More specifically, SCHEMEQ bears enough resemblance to the KSW construction such that it is possible to reuse KSW's proof on plaintext privacy in a blackbox fashion.
2. Next, we show in Section 5.7.4 that our main construction is computationally indistinguishable from an alternative scheme (referred to as SCHEMESYM), where the tokens and ciphertexts are symmetrically formed. As SCHEMESYM and SCHEMEREAL are computationally indistinguishable, it suffices to prove the ciphertext and query privacy in SCHEMESYM.

   The plaintext privacy of SCHEMESYM follows from the plaintext privacy of SCHEMEREAL. Since tokens and ciphertexts are symmetrically formed in SCHEMESYM, the tokens must be secure as well in SCHEMESYM.

5.7.3 Plaintext privacy of SCHEMEREAL

Lemma 5.7.5 (Selective plaintext privacy of SCHEMEREAL) *Assuming the generalized C3DH assumption and Assumption 1,* SCHEMEREAL *has selective plaintext privacy.*

We know that the KSW construction has plaintext privacy (in the public key setting). To prove the plaintext privacy of our construction, SCHEMEREAL, first observe the differences between our construction and the KSW construction.

1. Our construction introduces the $\bar{h}_{1,i}, \bar{h}_{2,i}$ terms. As a result, we need one extra group element for both the ciphertext and token: the C_0 and K terms in KSW become $C_0, C_\emptyset$ and $K_0, K_\emptyset$ in our construction.
2. Our construction removes the $\mathbb{G}_q$ elements from the K_0 and $K_\emptyset$ terms in the token. (To compare, observe the $Q_6 \xleftarrow{R} \mathbb{G}_q$ element in the K term of the KSW construction.)

The intuition behind the following proof (of Lemma 5.7.5) is to show that these modifications preserve the plaintext privacy of the KSW construction.

The proof of Lemma 5.7.5 consists of two parts:

1. We first add back the random hiding factors from $\mathbb{G}_q$ to the K_0 and $K_\emptyset$ terms in the token. The resulting scheme is called SCHEMEQ. We show that SCHEMEREAL and SCHEMEQ are computationally indistinguishable.
2. We prove the plaintext privacy of SCHEMEQ. The proof is a reduction showing that if there exists a polynomial-time adversary $\mathcal{A}$ that can break the plaintext privacy of SCHEMEQ, we can then build a polynomial-time simulator $\mathcal{B}$ that leverages the adversary $\mathcal{A}$, and breaks the plaintext privacy of the KSW construction.

Definition 5.7.6 (SCHEMEQ) *We add random hiding factors from the $\mathbb{G}_q$ subgroup to the terms*

K_0 *and* $K_\emptyset$ *in* SCHEMEREAL. *The resulting scheme is called* SCHEMEQ. *Below is a formal description of* SCHEMEQ.

For the readers' convenience, in the expression for the token TK below, we underline the parts where SCHEMEQ and SCHEMEREAL differ.

$Setup(1^\lambda)$: Same as the $Setup$ algorithm of SCHEMEREAL.

$Encrypt(\mathsf{MSK}, \vec{x})$: Same as the $Encrypt$ algorithm of SCHEMEREAL.

$GenToken(\mathsf{MSK}, \vec{v})$: Let $\vec{v} = (v_1, v_2, \ldots, v_n) \in (\mathbb{Z}_N)^n$. The $GenToken$ algorithm picks random exponents $f_1, f_2, \{r_{1,i}, r_{2,i}\}_{i=1}^n$ from $\mathbb{Z}_N$. Then, it chooses random hiding factors $R_0, R_\emptyset$ from the subgroup $\mathbb{G}_r$; random $Q_0, Q_\emptyset$ from $\mathbb{G}_q$; and random $\{\widehat{R}_{1,i}, \widehat{R}_{2,i}\}_{i=1}^n$ from $\mathbb{G}_{\widehat{r}}$.

Next, the $GenToken$ algorithm computes the following token:

$$\mathsf{TK} = \left(\begin{array}{l} K_0 = \underline{Q_0} R_0 \cdot \prod_{i=1}^n h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \quad K_\emptyset = \underline{Q_\emptyset} R_\emptyset \cdot \prod_{i=1}^n \bar{h}_{1,i}^{-r_{1,i}} \bar{h}_{2,i}^{-r_{2,i}} \\ \left\{ K_{1,i} = g_p^{r_{1,i}} g_q^{f_1 v_i} \widehat{R}_{1,i}, \;\; K_{2,i} = g_p^{r_{2,i}} g_q^{f_2 v_i} \widehat{R}_{2,i} \right\}_{i=1}^n \end{array} \right)$$

$Query(\mathsf{TK}_{\vec{v}}, \mathsf{CT}_{\vec{x}})$: Same as the $Query$ algorithm of SCHEMEREAL.

To reiterate, the underlined parts represent the places where SCHEMEQ and SCHEMEREAL differ.

Computational indistinguishability of SCHEMEREAL and SCHEMEQ

To show that SCHEMEREAL and SCHEMEQ are computationally indistinguishable, we further introduce a sequence of hybrid schemes:

$$\text{SCHEMEREAL} \overset{f_2=\omega f_1}{\Longrightarrow} \text{SCHEME1} \overset{Q_0}{\Longrightarrow} \text{SCHEME2} \overset{Q_\emptyset}{\Longrightarrow} \text{SCHEME3} \overset{\text{restore } f_1, f_2}{\Longrightarrow} \text{SCHEMEQ}$$

In the above, the text on top of the arrow highlights the modification we make to the former scheme to obtain the latter. These modifications will be explained in detail shortly when we formally define each hybrid scheme. We show that any two consecutive scheme in the above sequence are computationally indistinguishable.

Definition 5.7.7 (SCHEME1) *We first make slight modifications to* SCHEMEREAL *and obtain a hybrid scheme called* SCHEME1. *In* SCHEME1, *instead of picking independent and fresh and independent random numbers* f_1 *and* f_2 *for each token, the* $GenToken$ *algorithm picks* f_1 *at random, and lets* $f_2 = \omega f_1$, *where* ω *is a random number in* $\mathbb{Z}_N$ *chosen during the* $Setup$ *stage; and is kept secret by the master key owner. More specifically, we formally define* SCHEME1 *as below:*

$Setup(1^\lambda)$: Same as the $Setup$ algorithm of SCHEMEREAL, except that now, we pick an additional random number $\omega \in \mathbb{Z}_N$, and add it to the secret key. In the mathematical expressions below, we underline the parts where SCHEME1 differ from SCHEMEREAL.

$$\mathsf{Pvk} = \left(\underline{\omega,} \quad g_p, \; g_q, \; g_r, \; \widehat{g}_r, \quad \left\{ h_{1,i}, h_{2,i}, \bar{h}_{1,i}, \bar{h}_{2,i} \right\}_{i=1}^n \right)$$

$Encrypt(\mathsf{MSK}, \vec{x})$: Same as the $Encrypt$ algorithm of SCHEMEREAL.

$GenToken(\mathsf{MSK}, \vec{v})$: Instead of picking $f_1, f_2 \in \mathbb{Z}_N$ independently at random, the $GenToken$ algorithm picks $f \in \mathbb{Z}_N$ at random, and lets $f_1 = f$, and $f_2 = \omega f$. The rest of the $GenToken$ algorithm is the same as that of SCHEMEREAL.

$$\mathsf{TK} = \left(\begin{array}{l} K_0 = R_0 \cdot \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \quad K_\emptyset = R_\emptyset \cdot \prod_{i=1}^{n} \bar{h}_{1,i}^{-r_{1,i}} \bar{h}_{2,i}^{-r_{2,i}} \\ \left\{ K_{1,i} = g_p^{r_{1,i}} \underline{g_q^{f v_i}} \widehat{R}_{1,i}, \;\; K_{2,i} = g_p^{r_{2,i}} \underline{g_q^{\omega f v_i}} \widehat{R}_{2,i} \right\}_{i=1}^{n} \end{array} \right)$$

$Query(\mathsf{TK}_{\vec{v}}, \mathsf{CT}_{\vec{x}})$: Same as the $Query$ algorithm in SCHEMEREAL.

Claim 5.7.8 (Computational indistinguishability of SCHEMEREAL and SCHEME1) *Under the generalized C3DH assumption,* SCHEME1 *is computationally indistinguishable from* SCHEMEREAL.

Proof: We can prove this claim based on the generalized C3DH assumption and a hybrid argument. Intuitively, Claim 5.7.8 relies on the following observation.

Observation 5.7.1 (ℓ-C3DH) *Define the following distribution:*

$$\begin{array}{l} u_1, u_2, \ldots, u_\ell \xleftarrow{R} \mathbb{G}_q, \\ \omega \xleftarrow{R} \mathbb{Z}_N, \\ \widehat{R}_1, \ldots, \widehat{R}_\ell, \widetilde{R}_1, \ldots, \widetilde{R}_\ell, \overline{R}_1, \ldots, \overline{R}_\ell \xleftarrow{R} \mathbb{G}_{\widehat{r}}, \\ Q_1, Q_2, \ldots, Q_\ell \xleftarrow{R} \mathbb{G}_q \end{array}$$

Suppose an adversary is given the generators of each subgroup:

$$g_p, g_q, g_r, \widehat{g}_r$$

Let b *denote a random coin flip. If* $\mathsf{b} = 0$*, the adversary is given the tuple*

$$(u_1 \widehat{R}_1, \ldots, u_\ell \widehat{R}_\ell, \; u_1^\omega \widetilde{R}_1, \ldots, u_\ell^\omega \widetilde{R}_\ell)$$

if $\mathsf{b} = 1$*, the adversary is given the tuple*

$$(u_1 \widehat{R}_1, \ldots, u_\ell \widehat{R}_\ell, \; Q_1 \overline{R}_1, \ldots, Q_\ell \overline{R}_\ell)$$

Suppose the adversary outputs a guess b' *of* b*. Denote the adversary's advantage as* $\mathbf{Adv}_{\mathcal{A}} := \left| \Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2} \right|$*. Then no polynomial-time adversary can win this ℓ-C3DH game with more than negligible advantage.*

This observation can be proven through the generalized C3DH assumption and a simple hybrid argument. Shi et al. [37] also used the ℓ-C3DH assumption as an intermediate assumption in their proofs.

It is not hard to see that the above observation leads to Claim 5.7.8. The proof can be done through a simple reduction argument. Basically, if there exists an adversary that can distinguish

between SCHEMEREAL and SCHEME1, we can leverage that adversary to build a simulator $\mathcal{B}$ that can win the above ℓ-C3DH game. The simulator $\mathcal{B}$ is randomly given one of the following two tuples:

$$(u_1\widehat{R}_1, \ldots, u_\ell\widehat{R}_\ell,\ U_1 = u_1^\omega\widetilde{R}_1, \ldots, U_\ell = u_\ell^\omega\widetilde{R}_\ell)$$

or

$$(u_1\widehat{R}_1, \ldots, u_\ell\widehat{R}_\ell,\ U_1 = Q_1\overline{R}_1, \ldots, U_\ell = Q_\ell\overline{R}_\ell)$$

Now the simulator tries to determine which case it is.

The simulator leverages a distinguishing adversary $\mathcal{A}$ that tries to distinguish SCHEMEREAL and SCHEME1. Suppose that the adversary makes ℓ token queries.

In the setup phase of the game, the simulator generates the secret key (without the ω term) and retains the secret key to itself. Clearly, the simulator can successfully generate secret key given the generators of the different subgroups.

In answer to the jth token query, simulator uses the terms $u_j\widehat{R}_j$ and U_j from the ℓ-C3DH instance to build the following token:

$$\mathsf{TK} = \left(\begin{array}{l} K_0 = R_0 \cdot \prod_{i=1}^n h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \quad K_\emptyset = R_\emptyset \cdot \prod_{i=1}^n \bar{h}_{1,i}^{-r_{1,i}} \bar{h}_{2,i}^{-r_{2,i}} \\ \left\{ K_{1,i} = g_p^{r_{1,i}} \underline{(u_j\widehat{R}_j)^{v_i}} \widehat{R}_{1,i},\ \ K_{2,i} = g_p^{r_{2,i}} \underline{U_j^{v_i}} \widehat{R}_{2,i} \right\}_{i=1}^n \end{array} \right)$$

Clearly, if $\mathsf{b} = 0$, then the tokens are formed as in SCHEME1; if $\mathsf{b} = 1$, the tokens are formed as in SCHEMEREAL.

If $\mathcal{A}$ outputs a guess of SCHEME1, the simulator outputs a guess $\mathsf{b}' = 0$; if $\mathcal{A}$ outputs a guess of SCHEMEREAL, the simulator outputs a guess of $\mathsf{b}' = 1$. In this way, if $\mathcal{A}$ has ϵ advantage in distinguishing SCHEMEREAL and SCHEME1, the simulator will have ϵ in winning the ℓ-C3DH game. ■

Remark 5.7.1 *To prove computational indistinguishability between* SCHEMEREAL *and* SCHEME1*, we rely on the $\mathbb{G}_{\widehat{r}}$ subgroup. This means that our proof that* SCHEMEREAL *is computationally indistinguishable from* SCHEMEQ *relies on the $\mathbb{G}_{\widehat{r}}$ subgroup. Therefore, although we are able to computationally remove the $\mathbb{G}_q$ subgroup from the K_0 and $K_\emptyset$ terms in the token, it does NOT imply that one can do the same thing for the KSW construction, as the KSW construction does not have the $\mathbb{G}_{\widehat{r}}$ subgroup. To reiterate, our proof does NOT imply that one can safely remove the $\mathbb{G}_q$ subgroup from the K term in the token of the KSW construction.*

Definition 5.7.9 (SCHEME2) *We further modify* SCHEME1*, and add a random element $Q_0 \in \mathbb{G}_q$ to the term K_0 in the token. The resulting scheme is referred to as* SCHEME2*, and is formally defined as below:*

$Setup(1^\lambda)$: Same as in SCHEME1.

$Encrypt(\mathsf{MSK}, \vec{x})$: Same as in SCHEME1.

$GenToken(\mathsf{MSK}, \vec{v})$: The $GenToken$ picks a random $Q_0 \in \mathbb{G}_q$, and multiplies Q_0 to K_0. Note that a fresh Q_0 is generated each time $GenToken$ is called. The rest of the $GenToken$ algorithm is the same as in SCHEME1. In the expression below, we underline the parts where SCHEME2 and SCHEME1 differ.

$$\mathsf{TK} = \left(\begin{array}{l} K_0 = \underline{Q_0} R_0 \cdot \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \quad K_\emptyset = R_\emptyset \cdot \prod_{i=1}^{n} \bar{h}_{1,i}^{-r_{1,i}} \bar{h}_{2,i}^{-r_{2,i}} \\ \left\{ K_{1,i} = g_p^{r_{1,i}} g_q^{f v_i} \widehat{R}_{1,i}, \;\; K_{2,i} = g_p^{r_{2,i}} g_q^{\omega f v_i} \widehat{R}_{2,i} \right\}_{i=1}^{n} \end{array} \right)$$

$Query(\mathsf{TK}_{\vec{v}}, \mathsf{CT}_{\vec{x}})$: Same as in SCHEME1.

Again, for clarity, we underline the places where SCHEME2 differs from SCHEME1.

Claim 5.7.10 (Computational indistinguishability of SCHEME1 and SCHEME2) *Assume that Assumption 1 of the KSW paper [28] holds in the bilinear group $\mathbb{G}$, then* SCHEME2 *is computationally indistinguishable from* SCHEME1.

To prove this lemma, we first review Assumption 1 as stated by Katz et al.. We assume that this assumption holds when $\mathbb{G}_p \times \mathbb{G}_q \times \mathbb{G}_r$ belongs to a larger group of order $N = pqr\widehat{r}$. We restate Assumption 1 in the context of the larger group.

Definition 5.7.11 (Assumption 1 of the KSW construction [28]) *Any polynomial-time adversary has a negligible advantage in the following experiment:*

Let $N = pqr\widehat{r}$, let $g_p, g_q, g_r, \widehat{g}_r$ be random generators of $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_{\widehat{r}}$ respectively. Pick the following numbers at random: $Q_1, Q_2, Q_3 \in \mathbb{G}_q$, $R_1, R_2, R_3 \in \mathbb{G}_r$, $a, b, s \in \mathbb{Z}_p$, and a random bit b. If $\mathsf{b} = 0$, $\gamma = 0$; else if $\mathsf{b} = 1$, γ is chosen at random from $\mathbb{Z}_N$. Give the adversary the description of the bilinear group $(N, \mathbb{G}, \mathbb{G}_T, \mathsf{e})$, and the following set of values. The adversary's task is to guess the bit b.

$$S = \left\{ g_p, \;\; g_r, \;\; \widehat{g}_r, \;\; g_q R_1, \;\; g_p^b, \;\; g_p^{b^2}, \;\; g_p^a g_q, \;\; g_p^{ab} Q_1, \;\; g_p^s, \;\; g_p^{bs} Q_2 R_2, \;\; T = g_p^{b^2 s} g_q^\gamma R_3 \right\} \qquad (5.5)$$

The adversary outputs a guess b' of the bit b; and its advantage is defined as

$$\mathbf{Adv}_{\mathcal{A}} = \left| \Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2} \right|$$

Assumption states that no polynomial-time adversary can win this game with more than negligible advantage.

In fact, Claim 5.7.10 relies on a weaker assumption than Assumption 1. Specifically, we only need to reveal to the adversary a subset $S' \subset S$.

$$S' = \left\{ g_p, \;\; g_r, \;\; \widehat{g}_r, \;\; g_q R_1, \;\; g_p^{b^2}, \;\; g_p^a g_q, \;\; g_p^s, \;\; T = g_p^{b^2 s} g_q^\gamma R_3 \right\}$$

Definition 5.7.12 (Assumption W) *Given the set S', no polynomial-time adversary can decide whether $\gamma = 0$ or $\gamma \stackrel{R}{\leftarrow} \mathbb{Z}_N$ with more than negligible advantage.*

Clearly, Assumption 1 implies Assumption W, that is, Assumption W is weaker than Assumption 1.

Proof of Claim 5.7.10: We build a simulator $\mathcal{B}$ that tries to break Assumption 1. The simulator utilizes an adversary $\mathcal{A}$ that tries to distinguish SCHEME1 from SCHEME2. If the adversary $\mathcal{A}$ has

advantage ϵ in distinguishing SCHEME1 from SCHEME2, then the simulator $\mathcal{B}$ has advantage ϵ in breaking Assumption W.

The simulator $\mathcal{B}$ is given an instance of Assumption W, and it plays the following distinguishing game with the adversary $\mathcal{A}$. The adversary makes queries for ciphertexts and tokens, and in answer to these queries, the simulator computes ciphertexts and tokens following a certain strategy. The resulting ciphertexts and tokens are distributed either according to SCHEME1 or according to SCHEME2. In particular, if the simulator is given $T = g_p^{b^2 s} R_3$ from the Assumption W instance, then the encryption scheme used would be identically distributed as SCHEME1; otherwise, if $T = g_p^{b^2 s} Q_3 R_3$, the encryption scheme used would be identically distributed as SCHEME2.

- **Setup.** The simulator is given an instance of Assumption W, and it uses this knowledge to create the following secret key:

$$\mathsf{Pvk} = \left(\omega,\ g_p,\ g_r,\ \widehat{g}_r,\quad \left\{h_{1,i} = (g_p^{b^2})^{\omega y_i},\ h_{2,i} = g_p^{z_i}(g_p^{b^2})^{-y_i},\ \bar{h}_{1,i} = g_p^{c_i},\ \bar{h}_{2,i} = g_p^{d_i}\right\}_{i=1}^{n}\right)$$

  where $\omega, \{z_i, y_i, c_i, d_i\}_{i=1}^n$ are random exponents from $\mathbb{Z}_N$. In the above Pvk, the following elements are inherited from the Assumption W instance: g_p, g_q, g_r, $\widehat{g}_r$ and $g_p^{b^2}$.

  Notice that the simulator does not know g_q, which ought to part of the secret key. We show that the simulator is still able to answer ciphertext queries and token queries from the adversary appropriately, in spite of not knowing g_q.

- **Ciphertext query.** In spite of not knowing g_q, the simulator is able to compute ciphertexts, as it knows $g_q R_1$ from the Assumption W instance, and a generator g_r of the subgroup $\mathbb{G}_r$.

- **Token query.** To answer a token query, the simulator picks random values r, f from $\mathbb{Z}_N$; random hiding factor $R_0, R_\emptyset$ from the subgroup $\mathbb{G}_r$; and random $\{\widehat{R}_{1,i}, \widehat{R}_{2,i}\}_{i=1}^n$ from $\mathbb{G}_{\widehat{r}}$.

  The simulator uses the following strategy to decide the values of $\{r_{1,i}, r_{2,i}\}_{i=1}^n$. First, the simulator picks random exponents $\{\tau_i, \overline{r}_{1,i}, \overline{r}_{2,i}\}_{i=1}^n$ from $\mathbb{Z}_N$. The simulator then implicitly sets the values of $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ to be the following, *without actually computing them*:

$$\begin{aligned} \forall i \in [n]: \quad r_{1,i} &= af v_i + \tau_i s + \overline{r}_{1,i} \\ r_{2,i} &= af\omega v_i + \overline{r}_{2,i} \end{aligned} \tag{5.6}$$

  Using the above implicit values for $\{r_{1,i}, r_{2,i}\}_{i=1}^n$, the simulator is able to compute a token as below:

$$\begin{aligned} \forall i \in [n]: \quad K_{1,i} &= \widehat{R}_{1,i} \cdot (g_p^a g_q)^{f v_i} (g_p^s)^{\tau_i} g_p^{\overline{r}_{1,i}} \\ K_{2,i} &= \widehat{R}_{2,i} \cdot (g_p^a g_q)^{\omega f v_i} g_p^{\overline{r}_{2,i}} \end{aligned} \tag{5.7}$$

  In addition,

$$K_0 = \quad R_0 \cdot \prod_{i=1}^{n} T^{-\omega y_i \tau_i} \cdot h_{1,i}^{-\overline{r}_{1,i}} \cdot (g_p^{-a} R_1)^{f\omega z_i v_i} \cdot h_{2,i}^{-\overline{r}_{2,i}} \tag{5.8}$$

$$K_\emptyset = \quad R_\emptyset \cdot \prod_{i=1}^{n} \left((g_p^{-a} R_1)^{-f v_i} (g_p^s)^{\tau_i} g_p^{\overline{r}_{1,i}}\right)^{-c_i} \left((g_q^{-a} R_1)^{\omega f v_i} g_p^{\overline{r}_{2,i}}\right)^{-d_i} \tag{5.9}$$

Notice that the above equations make use of the term $g_p^{-a} R_1$. This can be obtained from the terms $g_q R_1$ and $g_p^a g_q$ inherited from the Assumption W instance:

$$g_p^{-a} R_1 = \frac{g_q R_1}{g_p^a g_q}$$

It is not hard to see that $\{K_{1,i}, K_{2,i}\}_{i=1}^n$ as defined in Equation (5.7), and $K_\emptyset$ as defined in Equation (5.9), are correctly formed as in SCHEME1 (or SCHEME2). Recall that the terms $K_\emptyset, \{K_{1,i}, K_{2,i}\}_{i=1}^n$ have the same form in both SCHEME1 and SCHEME2. It remains to verify that K_0, as defined in Equation (5.8), is distributed either as in SCHEME1 or as in SCHEME2, depending on the value of γ from the Assumption W instance.

Observation 5.7.2 *If γ from the Assumption W instance is equal to 0, then K_0 as defined in Equation (5.8) is distributed as in* SCHEME1. *Otherwise, if $\gamma \xleftarrow{R} \mathbb{Z}_N$, K_0 as defined in Equation (5.8) is distributed as in* SCHEME2.

To see why K_0 follows the correct distribution, let $K_{0,p}, K_{0,q}, K_{0,r}$ denote the projections of K_0 into the subgroups $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r$ respectively. Clearly, $K_{0,r}$ has the correct distribution.

We now verify that $K_{0,q}$ and $K_{0,p}$ have the correct distribution.

It is not hard to see that

$$K_{0,q} = g_q^{-\gamma\kappa} \qquad \text{where } \kappa = \omega \sum_{i=1}^{n} y_i \tau_i$$

Clearly, if $\gamma = 0$ in the Assumption W instance, then $K_{0,q}$ is distributed as in SCHEME1, i.e., K_0 does not contain an element from the subgroup $\mathbb{G}_q$. We now need to show that if $\gamma \xleftarrow{R} \mathbb{Z}_N$, $K_{0,q}$ is distributed as in SCHEME2, that is, K_0 contains a random element from the subgroup $\mathbb{G}_q$. To prove this, it suffices to observe that κ is distributed uniformly at random in $\mathbb{Z}_N$, and is independent of $\{r_{1,i}, r_{2,i}\}_{i=1}^n$.

Remark 5.7.2 *In fact, it suffices to pick $\tau_1 \xleftarrow{R} \mathbb{Z}_N$, and fix $\tau_i = 0$ for $i \in [2, n]$.*

It remains to verify that $K_{0,p}$ has the correct distribution. The correct distribution of $K_{0,p}$ should be:

$$\begin{aligned} \textstyle\prod_{i=1}^n h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}} &= \textstyle\prod_{i=1}^n (g_p^{-b^2 \omega y_i})^{af v_i + \tau_i s + \overline{r}_{1,i}} \cdot (g_p^{-z_i} g_p^{b^2 y_i})^{af\omega v_i + \overline{r}_{2,i}} \\ &= \textstyle\prod_{i=1}^n g_p^{-b^2 s\omega y_i \tau_i} h_{1,i}^{-\overline{r}_{1,i}} g_p^{-af\omega z_i v_i} h_{2,i}^{-\overline{r}_{2,i}} \end{aligned} \tag{5.10}$$

It is not very hard to see that the K_0 defined in Equation (5.8) has the same $\mathbb{G}_p$ component as the above Equation (5.10). A crucial observation here is that all terms involving $g_p^{ab^2}$ (which is unknown to the simulator) cancel out. This is the reason why the simulator can generate the token efficiently.

- **Guess.** The simulator $\mathcal{B}$ outputs the same guess b' output by the adversary $\mathcal{A}$.

Clearly, if the adversary $\mathcal{A}$ has advantage ϵ in distinguishing SCHEME1 and SCHEME2, then the simulator $\mathcal{B}$ also has advantage ϵ in breaking Assumption W. This completes the proof of Claim 5.7.10. ■

Definition 5.7.13 (SCHEME3) *We further modify* SCHEME2*, and add a random element $Q_\emptyset \in \mathbb{G}_q$ to the term $K_\emptyset$ in the token. In the resulting scheme (referred to as* SCHEME3*), both the terms K_0 and $K_\emptyset$ has a random element from the $\mathbb{G}_q$ subgroup.* SCHEME3 *is formally defined as below:*

$Setup(1^\lambda)$: Same as in SCHEME2.

$Encrypt(\mathsf{MSK}, \vec{x})$: Same as in SCHEME2.

$GenToken(\mathsf{MSK}, \vec{v})$: The $GenToken$ picks a random $Q_0, Q_\emptyset \in \mathbb{G}_q$, and multiplies Q_0 and $Q_\emptyset$ to K_0 and $K_\emptyset$ respectively. $GenToken$ algorithm is the same as in SCHEME1. In the expression below, we underline the parts where SCHEME2 and SCHEME3 differ.

$$\mathsf{TK} = \left(\begin{array}{l} K_0 = Q_0 R_0 \cdot \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \quad K_\emptyset = \underline{Q_\emptyset} R_\emptyset \cdot \prod_{i=1}^{n} \bar{h}_{1,i}^{-r_{1,i}} \bar{h}_{2,i}^{-r_{2,i}} \\ \left\{ K_{1,i} = g_p^{r_{1,i}} g_q^{f v_i} \widehat{R}_{1,i}, \;\; K_{2,i} = g_p^{r_{2,i}} g_q^{\omega f v_i} \widehat{R}_{2,i} \right\}_{i=1}^{n} \end{array} \right)$$

$Query(\mathsf{TK}_{\vec{v}}, \mathsf{CT}_{\vec{x}})$: Same as in SCHEME2.

Claim 5.7.14 (Computational indistinguishability of SCHEME2 and SCHEME3) *Given that Assumption 1 of the KSW paper [28] holds in the bilinear group $\mathbb{G}$, then* SCHEME3 *is computationally indistinguishable from* SCHEME2*.*

Proof: The proof of this claim is very similar to that of Claim 5.7.10. The only difference is that in this proof, the simulator needs to rerandomize the term K_0 with a random element from the subgroup $\mathbb{G}_{qr} = \mathbb{G}_q \times \mathbb{G}_r$. This can be achieved since the simulator knows the terms $g_q R_1$ and g_r. (In comparison, in the proof of Claim 5.7.10, the simulator rerandomizes the term $K_\emptyset$ with an element from $\mathbb{G}_r$). ■

Recall that we are trying to show that SCHEMEREAL and SCHEMEQ are computationally indistinguishable. We have made a sequence of modifications to SCHEMEREAL, and have obtained SCHEME3. So far, we have shown that SCHEMEREAL and SCHEME3 are computationally indistinguishable. We now further modify SCHEME3 and finally obtain SCHEMEQ. In SCHEME1, SCHEME2 and SCHEME3, the f_1 and f_2 exponents in the tokens satisfy the relation $f_2 = \omega f_1$, where ω is a pre-determined secret. We now restore the f_1 and f_2 exponents as independent fresh random numbers.

Claim 5.7.15 *Assuming the generalized C3DH assumption,* SCHEME3 *and* SCHEMEQ *are computationally indistinguishable.*

Proof: Similar to that of Claim 5.7.8. ■

Plaintext privacy of SCHEMEQ

We have shown that SCHEMEREAL is computationally indistinguishable from SCHEMEQ. We now show that SCHEMEQ has selective plaintext privacy. This implies that SCHEMEREAL has selective plaintext privacy as well.

The original KSW construction runs in a bilinear group of order $N = pqr$. This part of the proof relies on the observation that if we run the KSW construction in the subgroup $\mathbb{G}_{pqr} = \mathbb{G}_p \times \mathbb{G}_q \times \mathbb{G}_r$ residing in a larger bilinear group of order $N = pqr\widehat{r}$, the KSW construction still has plaintext privacy. Fundamentally, this relies on the fact that Assumption 1 still holds when the bilinear group $\mathbb{G}_{pqr}$ in question resides in the context of a larger group.

Lemma 5.7.16 *Suppose that Assumption 1 holds in the bilinear group $\mathbb{G}$, then* SCHEMEQ *has selective plaintext privacy.*

Proof: The proof is based on the selective plaintext privacy of the KSW construction. We show that if there exists a polynomial-time adversary $\mathcal{A}$ that can break the selective plaintext privacy of SCHEMEQ, we can build a polynomial-time simulator $\mathcal{B}$ that leverages $\mathcal{A}$ to break the selective plaintext privacy of the KSW construction. Recall that the KSW construction uses a bilinear group of order $N = pqr$. We assume that this group resides in a larger group of size $N = pqr\widehat{r}$, and that Assumption 1 still holds in the context of this larger group.

The simulator $\mathcal{B}$ acts two different roles. On one hand, it interacts with a KSW challenger $\mathcal{C}$, and tries to break the selective plaintext privacy of KSW. On the other hand, it acts as a challenger to the SCHEMEQ adversary $\mathcal{A}$. In essence, the simulator $\mathcal{B}$ uses the following strategy to interact with $\mathcal{A}$: whenever $\mathcal{A}$ submits a ciphertext or token query, the simulator $\mathcal{B}$ simply forwards it along to the challenger $\mathcal{C}$. In return, $\mathcal{B}$ obtains a KSW ciphertext or token. Now $\mathcal{B}$ augments the KSW ciphertext or token before handing the answer over to the adversary $\mathcal{A}$. For example, part of the augmentation performed by $\mathcal{B}$ is to fill in the terms $C_\emptyset$ and $K_\emptyset$.

- **Init.** The SCHEMEQ adversary $\mathcal{A}$ commits to a ciphertext challenge $(\vec{x}_0, \vec{x}_1)$ to the simulator $\mathcal{B}$. $\mathcal{B}$ forwards the same challenge $(\vec{x}_0, \vec{x}_1)$ to $\mathcal{C}$.
- **Setup.** $\mathcal{C}$ runs the $Setup$ algorithm of KSW, and gives the following public key to the simulator $\mathcal{B}$.
$$\mathsf{PK} = (g_p,\ g_r,\ \widehat{g}_r,\ \ Q = g_q \cdot R_0, \quad \{H_{1,i}, H_{2,i}\}_{i=1}^n)$$
In addition, $\mathcal{B}$ generates the following secrets:
$$\{\bar{h}_{1,i} = g_p^{y_i},\ \bar{h}_{1,i} = g_p^{z_i}\}_{i=1}^n$$
where $\{y_i, z_i\}_{i=1}^n$ are random numbers from $\mathbb{Z}_N$.
- **Ciphertext query.** Whenever the adversary $\mathcal{A}$ submits a ciphertext query for the vector $\vec{x} \in (\mathbb{Z}_N)^n$, $\mathcal{B}$ computes the following ciphertext and returns it to the adversary. Pick random exponents s, t, α, β from $\mathbb{Z}_N$; random hiding factors $\widehat{R}_0, \widehat{R}_\emptyset$ from the subgroup $\mathbb{G}_{\widehat{r}}$; and random $\{R_{1,i}, R_{2,i}\}_{i=1}^n$ from $\mathbb{G}_r$.
$$\mathsf{CT} = \left(\begin{array}{l} C_0 = \widehat{R}_0 \cdot g_p^s, \quad C_\emptyset = \widehat{R}_\emptyset \cdot g_p^t \\ \left\{C_{1,i} = H_{1,i}^s \bar{h}_{1,i}^t Q^{\alpha x_i} R_{1,i},\ \ C_{2,i} = H_{2,i}^s \bar{h}_{2,i}^t Q^{\beta x_i} R_{2,i}\right\}_{i=1}^n \end{array} \right)$$
- **Token query.** Suppose that the adversary $\mathcal{A}$ makes a token query for the vector $\vec{v} \in (\mathbb{Z}_N)^n$. The simulator asks $\mathcal{C}$ to generate a KSW token for the same vector $\vec{v}$. Suppose the KSW token for $\vec{v}$ is formed as below:
$$\mathsf{KSW.TK} = (k_0,\ \{k_{1,i}, k_{2_i}\}_{i=1}^n)$$

The simulator now transforms this KSW token into a SCHEMEQ token as below. The simulator picks a random exponent $r \xleftarrow{R} \mathbb{Z}_N$; a random hiding factor $R_\emptyset$ from the subgroup $\mathbb{G}_r$; and random $\{\widehat{R}_{1,i}, \widehat{R}_{2,i}\}_{i=1}^n$ from $\mathbb{G}_{\widehat{r}}$.

$$\mathsf{TK} = \left(\begin{array}{l} K_0 = k_0, \quad K_\emptyset = R_\emptyset Q^r \prod_{i=1}^n k_{1,i}^{-y_i} k_{2_i}^{-z_i} \\ \left\{ K_{1,i} = k_{1,i} \widehat{R}_{1,i}, \;\; K_{2,i} = k_{2,i} \widehat{R}_{2,i} \right\}_{i=1}^n \end{array} \right)$$

- **Challenge.** The adversary $\mathcal{A}$ submits a ciphertext challenge for the vector $\vec{x} \in (\mathbb{Z}_N)^n$. The simulator $\mathcal{B}$ forwards the challenge to the KSW challenger $\mathcal{C}$. As a result, $\mathcal{B}$ obtains the following KSW challenge ciphertext from $\mathcal{C}$:

$$\mathsf{KSW.CT} = (c_0, \;\; \{c_{1,i}, c_{2,i}\}_{i=1}^n)$$

  The simulator transforms the above KSW ciphertext to a ciphertext under SCHEMEQ. It picks $t \xleftarrow{R} \mathbb{Z}_N$, $\widehat{R}_0, \widehat{R}_\emptyset \xleftarrow{R} \mathbb{G}_{\widehat{r}}$, and computes:

$$\mathsf{CT} = \left(\begin{array}{l} C_0 = \widehat{R}_0 \cdot c_0, \quad C_\emptyset = \widehat{R}_\emptyset \cdot g_p^t \\ \left\{ C_{1,i} = c_{1,i} \bar{h}_{1,i}^t, \;\; C_{2,i} = c_{2,i} \bar{h}_{2,i}^t \right\}_{i=1}^n \end{array} \right)$$

- **More ciphertext and token queries.** Same as above.
- **Guess.** The simulator $\mathcal{B}$ outputs the same guess as the adversary $\mathcal{A}$.

It is not hard to verify that in the above simulation, the ciphertexts and tokens computed by $\mathcal{B}$ has the correct distribution. Clearly, if $\mathcal{A}$ has ϵ advantage in breaking SCHEMEQ, then the simulator $\mathcal{B}$ has ϵ advantage in breaking KSW. This completes the proof of Lemma 5.7.16. ■

5.7.4 Indistinguishability of SCHEMEREAL and SCHEMESYM

We now show that SCHEMEREAL is computationally indistinguishable from a scheme called SCHEMESYM, where the tokens and the ciphertexts are symmetrically formed. The proof is carried out in the following two steps:

1. We first define SCHEMESYM, and show that SCHEMEREAL is computationally indistinguishable from SCHEMESYM.
2. Next, we show that in SCHEMESYM, the tokens and ciphertexts are symmetrically formed.

SCHEMESYM

We make modifications to SCHEMEREAL, and obtain a new scheme called SCHEMESYM. In short, we modify the way the $GenToken$ algorithm picks the exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$. In SCHEMESYM, the exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ are no longer picked completely at random from $\mathbb{Z}_p$. Instead, these exponents are now picked at random from a two-dimensional subspace of the vector space $\mathbb{F}_p^{2n}$.

Definition 5.7.17 (SCHEMESYM) *We make the following modifications to* SCHEMEREAL*, and the resulting scheme is called* SCHEMESYM.

$Setup(1^\lambda)$: The setup algorithm first chooses a secret key as in SCHEMEREAL. Additionally, it chooses the following random exponents from $\mathbb{Z}_p$, and keeps them secret.

$$\{y_{1,i},\ z_{1,i},\ y_{2,i},\ z_{2,i}\}_{i=1}^{n}$$

$Encrypt(\mathsf{MSK}, \vec{x})$: Same as the $Encrypt$ algorithm of SCHEMEREAL.

$GenToken(\mathsf{MSK}, \vec{v})$: Instead of picking $\{r_{1,i}, r_{2,i}\}_{i=1}^{n}$ independently at random from $\mathbb{Z}_p$, the $GenToken$ picks two random numbers $\rho, \tau \xleftarrow{R} \mathbb{Z}_p$, and sets the values of $\{r_{1,i}, r_{2,i}\}_{i=1}^{n}$ as below:

$$\forall i \in [n]: \quad r_{1,i} = \rho y_{1,i} + \tau z_{1,i}$$
$$r_{2,i} = \rho y_{2,i} + \tau z_{2,i}$$

The rest of the $GenToken$ proceeds as in SCHEMEREAL.

$Query(\mathsf{TK}_{\vec{v}}, \mathsf{CT}_{\vec{x}})$: The same as the $Query$ algorithm in SCHEMEREAL.

One way to understand the above construction SCHEMESYM is as follows. Let $\vec{y} = \{y_{1,i}, y_{2,i}\}_{i=1}^{n}$, let $\vec{z} = \{z_{1,i}, z_{2,i}\}_{i=1}^{n}$, let $\vec{r} = \{r_{1,i}, r_{2,i}\}_{i=1}^{n}$. It is not hard to see that $\vec{r}$ is chosen at random from a 2-dimensional subspace generated by $\vec{y}$ and $\vec{z}$. Essentially, SCHEMESYM always chooses a 2-dimensional subspace during the setup phase. Later, when constructing tokens, SCHEMESYM always picks the exponents $\vec{r}$ at random from this prescribed 2-dimensional subspace. Due to the Decisional Linear assumption, picking the exponents from a 2-dimensional subspace is computationally indistinguishable from picking the exponents completely at random from the entire vector space $\mathbb{F}_p^{2n}$. We state this intuition in the following lemma.

So far, it may not be entirely clear why the ciphertexts and tokens are symmetrically formed in SCHEMESYM. We explain why this is the case in Section 5.7.4.

Lemma 5.7.18 *Assume that the D-Linear assumption holds in $\mathbb{G}_p$,* SCHEMESYM *is computationally indistinguishable from* SCHEMEREAL.

Informally, the above Lemma 5.7.18 relies on the following observation.

Observation 5.7.3 (ℓ-DLinear) *Let ℓ be an integer greater than 2. Suppose a challenger picks two random vectors $\vec{y} = (y_1, y_2, \ldots, y_\ell) \xleftarrow{R} \mathbb{F}_p^\ell$, and $\vec{z} = (z_1, z_2, \ldots, z_\ell) \xleftarrow{R} \mathbb{F}_p^\ell$. The challenger then flips a random coin b, and generates a random vector $\vec{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_\ell)$ in one of the following ways, depending on the outcome of the coin flip b:*

- *If $\mathsf{b} = 0$, the challenger picks $\gamma_1, \gamma_2, \ldots, \gamma_\ell$ independently at random from $\mathbb{Z}_p$. In other words, the vector $\vec{\gamma}$ is picked at random from the vector space $\mathbb{F}_p^\ell$.*
- *If $\mathsf{b} = 1$, the challenger picks the vector $\vec{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_\ell)$ from the 2-dimensional subspace[1] generated by $\vec{y}, \vec{z}$. Let $\mathsf{closure}(\vec{y}, \vec{z})$ denote the subspace in $\mathbb{F}_p^\ell$ generated by $\vec{y}$ and $\vec{z}$. The following algorithm allows the challenger to pick a random vector $\vec{\gamma}$ from $\mathsf{closure}(\vec{y}, \vec{z})$. Pick $s, t \xleftarrow{R} \mathbb{Z}_p$, and compute*

$$\vec{\gamma} = s\vec{y} + t\vec{z}$$

[1]In the unlikely event that $\vec{y}$ and $\vec{z}$ are linearly dependent, $\dim(\mathsf{closure}(\vec{y}, \vec{z})) < 2$. However, this happens with negligible probability.

Define the following notation:

$$g_p^{\vec{x}} := \left(g_p^{x_1},\ g_p^{x_2},\ \ldots,\ g_p^{x_\ell}\right) \qquad \textit{where } \vec{x} \in \mathbb{F}_p^\ell$$

Now the challenger gives an adversary the description of the group, $(N = pqr\widehat{r},\ \mathbb{G},\ \mathbb{G}_T,\ \mathsf{e})$*, generators of each subgroup,* $g_p, g_q, g_r, \widehat{g}_r$*, and the following tuple:*

$$\left(g_p^{\vec{y}},\ \ g_p^{\vec{z}},\ \ g_p^{\vec{\gamma}}\right)$$

The adversary's task is to guess the outcome of the coin flip b*. We claim that no polynomial-time adversary is able to guess the outcome of the coin flip* b *with more than negligible advantage. In fact, we show that this problem is at least as hard as the D-Linear problem.*

Proof: We use a hybrid argument to show that the ℓ-DLinear problem (Observation 5.7.3) is at least as hard as the D-Linear problem. We first review the D-Linear assumption. Suppose an adversary is given a description of the group, $(N = pqr\widehat{r},\ \mathbb{G},\ \mathbb{G}_T,\ \mathsf{e})$, generators of each subgroup, $g_p, g_q, g_r, \widehat{g}_r$, and the following tuple:

$$\left(g_p,\ g_p^a,\ g_p^b,\ g_p^{a\rho},\ g_p^{b\tau},\ Y\right)$$

where a, b, ρ, τ are random exponents in $\mathbb{Z}_p$. The adversary tries to decide whether $Y = g_p^{\rho+\tau}$ or whether Y is a random number in $\mathbb{G}_p$. The D-Linear assumption states that no polynomial-time adversary can have more than negligible advantage in this experiment.

We now prove Observation 5.7.3 (the ℓ-DLinear assumption) through a hybrid argument. We define the following sequence of games, where $*$ represents a random number from the group $\mathbb{G}_p$.

Game	What the challenger gives to the adversary
Game_ℓ	$(g_p^{\vec{y}},\ g_p^{\vec{z}},\ g_p^{sy_1+tz_1},\ g_p^{sy_2+tz_2},\ g_p^{sy_3+tz_3},\ \ldots,\ g_p^{sy_{\ell-2}+tz_{\ell-2}},\ g_p^{sy_{\ell-1}+tz_{\ell-1}},\ g_p^{sy_\ell+tz_\ell})$
$\text{Game}_{\ell-1}$	$(g_p^{\vec{y}},\ g_p^{\vec{z}},\ g_p^{sy_1+tz_1},\ g_p^{sy_2+tz_2},\ g_p^{sy_3+tz_3},\ \ldots,\ g_p^{sy_{\ell-2}+tz_{\ell-2}},\ g_p^{sy_{\ell-1}+tz_{\ell-1}},\ *)$
$\text{Game}_{\ell-2}$	$(g_p^{\vec{y}},\ g_p^{\vec{z}},\ g_p^{sy_1+tz_1},\ g_p^{sy_2+tz_2},\ g_p^{sy_3+tz_3},\ \ldots,\ g_p^{sy_{\ell-2}+tz_{\ell-2}},\ *,\ *)$
$\ldots$	$\ldots$
Game_2	$(g_p^{\vec{y}},\ g_p^{\vec{z}},\ g_p^{sy_1+tz_1},\ g_p^{sy_2+tz_2},\ *,\ \ldots,\ *,\ *,\ *)$

It is not hard to see that Game_ℓ is equivalent to the ℓ-DLinear experiment when $\mathsf{b} = 1$; and Game_2 is equivalent to the ℓ-DLinear experiment when $\mathsf{b} = 0$. Due to the hybrid argument, it suffices to prove that no polynomial-time adversary can distinguish between two adjacent games.

We now show that if there exists a polynomial-time adversary $\mathcal{A}$ that can distinguish between two adjacent games Game_d and Game_{d-1} with ϵ advantage, then we can build a polynomial-time simulator $\mathcal{B}$ that utilizes $\mathcal{A}$ as a black box, and wins the D-Linear experiment also with ϵ advantage. We now explain how the simulator $\mathcal{B}$ works.

Suppose $\mathcal{B}$ is given the D-Linear instance $(g_p,\ g_p^a,\ g_p^b,\ g_p^{a\rho},\ g_p^{b\tau},\ Y)$, and tries to decide whether $Y = g_p^{\rho+\tau}$ or $Y \stackrel{R}{\leftarrow} \mathbb{G}_p$. The simulator picks random elements k_2, k_3, …, k_{d-1}, y_d, y_{d+1}, …, y_ℓ, and w_2, w_3, …, w_{d-1}, z_d, z_{d+1}, …, z_ℓ from $\mathbb{Z}_N$, and implicitly sets:

$$\vec{y} = (a,\ k_2 a,\ k_3 a,\ \ldots,\ k_{d-1}a,\ \ y_d,\ y_{d+1},\ \ldots,\ y_\ell)$$

$$\vec{z} = (b,\ w_2 b,\ w_3 b,\ \ldots,\ w_{d-1} b,\ \ z_d,\ z_{d+1},\ \ldots,\ z_\ell)$$

It also implicitly sets

$$s = \rho y_d^{-1}, \quad t = \tau z_d^{-1}$$

where multiplicative inverses are taken modular N. (For our purposes, this is equivalent to taking multiplicative inverses modular p.)

Note that the simulator does not know the values of a, b, ρ, τ. It merely sets the above parameters implicitly, without actually computing them.

Now the simulator $\mathcal{B}$ gives the adversary $\mathcal{A}$ the following tuple:

$$g_p^{\vec{y}},\ g_p^{\vec{z}},\ \ (g_p^{a\rho})^{y_d^{-1}} (g_p^{b\tau})^{{z_d}^{-1}},\ \ \left\{ (g_p^{a\rho})^{k_i y_d^{-1}} (g_p^{b\tau})^{w_i z_d^{-1}} \right\}_{i=2}^{d-1},\ \ Y,\ *,\ *,\ \ldots,\ *$$

Clearly, if $Y = g_p^{\rho+\tau}$, then the above experiment is identically distributed as Game_d. Otherwise, if Y is a random element in $\mathbb{G}_p$, then the above experiment is identically distributed as Game_{d-1}. Hence, if $\mathcal{A}$ can distinguish between Game_{d-1} and Game_d with ϵ advantage, then $\mathcal{B}$ can win the D-Linear experiment with ϵ advantage as well. ■

Given the ℓ-DLinear assumption, which is implied by the Decisional Linear assumption, we proceed to prove Lemma 5.7.18, that is, SCHEMEREAL is computationally indistinguishable from SCHEMESYM.

Proof of Lemma 5.7.18: Let $\ell = 2n$. We show that distinguishing between SCHEMEREAL and SCHEMESYM is at least as hard as the ℓ-DLinear problem as stated in Observation 5.7.3. Our proof relies a hybrid argument on the number of token queries made by the adversary. Let k denote the number of token queries made by the adversary. We define a sequence of games, $\text{Game}_0, \text{Game}_1, \ldots, \text{Game}_k$. In Game_d ($0 \le d \le k$), for the first d tokens queried, the challenger picks the exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ from a pre-determined 2-dimensional subspace; and for the remaining token queries $d+1, \ldots, k$, the challenger picks completely random exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ from $\mathbb{F}_p^{2n}$.

More specifically, Game_d ($0 \le d \le k$) is formally defined as below.

- **Setup.** The challenger picks two random vectors

$$\vec{y} = \{y_{1,i},\ y_{2,i}\}_{i=1}^n \ \xleftarrow{R}\ \mathbb{F}_p^{2n}$$
$$\vec{z} = \{z_{1,i},\ z_{2,i}\}_{i=1}^n \ \xleftarrow{R}\ \mathbb{F}_p^{2n}$$

  and keeps them secret. These two vectors determine a 2-dimensional subspace $\mathsf{closure}(\vec{y}, \vec{z})$. Later, when the challenger answers the first d token queries made by the adversary, it will pick the exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ at random from this subspace. The challenger now calls the $Setup$ algorithm to generate a secret key as in SCHEMEREAL.

- **Ciphertext queries.** The challenger answers all ciphertext queries by directly calling the $Encrypt$ algorithm.

- **Token queries.** For the first d token queries, the challenger picks exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ as below. Pick two random numbers $\rho, \tau \xleftarrow{R} \mathbb{Z}_p$, and sets the values of $\vec{r} := \{r_{1,i}, r_{2,i}\}_{i=1}^n$ to be

the following:

$$\forall i \in [n]: \quad r_{1,i} = \rho y_{1,i} + \tau z_{1,i}$$
$$r_{2,i} = \rho y_{2,i} + \tau z_{2,i}$$

Expressed in the vector form,

$$\vec{r} = \rho \vec{y} + \tau \vec{z}$$

In other words, $\vec{r}$ is picked at random from the 2-dimensional subspace $\mathsf{closure}(\vec{y}, \vec{z})$.

For the remaining token queries $d+1, \ldots, k$, the challenger generates tokens normally by calling the $GenToken$ algorithm.

It is not hard to see that Game_0 is identically distributed as SCHEMEREAL, and Game_k is identically distributed as SCHEMESYM. Due to the hybrid argument, it suffices to show that no polynomial-time adversary is able to distinguish between two adjacent games Game_{d-1} and Game_d ($1 \leq d \leq k$) with more than negligible advantage.

We now show that if there exists a polynomial-time adversary $\mathcal{A}$ that can distinguish between Game_{d-1} and Game_d ($1 \leq d \leq k$) with ϵ advantage, we can build a polynomial-time simulator $\mathcal{B}$ that uses $\mathcal{A}$ as a blackbox, and breaks the ℓ-DLinear assumption also with ϵ advantage. We now explain how the simulator $\mathcal{B}$ works. Suppose $\mathcal{B}$ is given the following ℓ-DLinear instance $(g_p^{\vec{y}}, g_p^{\vec{z}}, g_p^{\vec{\gamma}})$, where $\vec{y} = \{y_{1,i}, y_{2,i}\}_{i=1}^n$, $\vec{z} = \{z_{1,i}, z_{2,i}\}_{i=1}^n$, and $\vec{\gamma} = \{\gamma_{1,i}, \gamma_{2,i}\}_{i=1}^n$. Now the simulator tries to distinguish whether $\vec{\gamma} \in \mathsf{closure}(\vec{y}, \vec{z})$, or whether $\vec{\gamma}$ is a random vector in $\mathbb{F}_p^{2n}$. To do this, the simulator will set the exponents $\vec{r} = \{r_{1,i}, r_{2,i}\}_{i=1}^n$ in the first $d-1$ tokens to be random vectors in $\mathsf{closure}(\vec{y}, \vec{z})$. The simulator sets the exponents $\vec{r}$ in the d^{th} to be the vector $\vec{\gamma}$. For the remaining token queries, the simulator chooses random exponents $\vec{r} \xleftarrow{R} \mathbb{F}_p^{2n}$. In this way, if $\vec{\gamma} \xleftarrow{R} \mathsf{closure}(\vec{y}, \vec{z})$, the simulation is equivalent to Game_d; otherwise, if $\vec{\gamma} \xleftarrow{R} \mathbb{F}_p^{2n}$, the simulation is equivalent to Game_{d-1}.

- **Setup.** The simulator picks the following secret key:

$$\mathsf{MSK} = (g_p,\ g_q,\ g_r,\ \widehat{g}_r,\ \left\{h_{1,i} = g_p^{\omega_{1,i}},\ h_{2,i} = g_p^{\omega_{2,i}},\ \bar{h}_{1,i} = g_p^{\kappa_{1,i}},\ \bar{h}_{2,i} = g_p^{\kappa_{2,i}}\right\})$$

  where $\omega_{1,i}, \omega_{2,i}, \kappa_{1,i}, \kappa_{2,i}$ are random exponents in $\mathbb{Z}_p$.

- **Ciphertext queries.** The simulator answers all ciphertext queries by directly calling the $Encrypt$ algorithm.

- **Token queries.** For all token queries, the simulator picks random exponents f_1, f_2 from $\mathbb{Z}_N$; random hiding factors $R_0, R_\emptyset$ from the subgroup $\mathbb{G}_r$; and random $\{\widehat{R}_{1,i}, \widehat{R}_{2,i}\}_{i=1}^n$ from $\mathbb{G}_{\widehat{r}}$. It chooses the values of $\vec{r} = \{r_{1,i}, r_{2,i}\}_{i=1}^n$ in one of the following ways:

  - For the first $d-1$ token queries, the challenger picks random ρ, τ from $\mathbb{Z}_p$, (ρ, τ are picked as fresh random numbers for each of the first $d-1$ token queries.) and implicitly lets $\vec{r} = \{r_{1,i}, r_{2,i}\}_{i=1}^n$ to be the following (without actually computing it):

$$\vec{r} = \rho \vec{y} + \tau \vec{z}$$

    In the above expression, $\vec{y}$ and $\vec{z}$ are inherited from the ℓ-DLinear instance. Note that the simulator does not know the values of $\vec{y}$ and $\vec{z}$, it implicitly sets the vector $\vec{r}$ without

computing its value. Now the simulator computes the following token:

$$\mathsf{TK} = \left(\begin{array}{l} K_0 = R_0 \cdot \prod_{i=1}^{n} (g_p^{y_{1,i}})^{-\rho\omega_{1,i}} (g_p^{z_{1,i}})^{-\tau\omega_{1,i}} (g_p^{y_{2,i}})^{-\rho\omega_{2,i}} (g_p^{z_{2,i}})^{-\tau\omega_{2,i}}, \\ K_\emptyset = R_\emptyset \cdot \prod_{i=1}^{n} (g_p^{y_{1,i}})^{-\rho\kappa_{1,i}} (g_p^{z_{1,i}})^{-\tau\kappa_{1,i}} (g_p^{y_{2,i}})^{-\rho\kappa_{2,i}} (g_p^{z_{2,i}})^{-\tau\kappa_{2,i}}, \\ \left\{ K_{1,i} = (g_p^{y_{1,i}})^{\rho} (g_p^{z_{1,i}})^{\tau} g_q^{f_1 v_i} \widehat{R}_{1,i}, \quad K_{2,i} = (g_p^{y_{2,i}})^{\rho} (g_p^{z_{2,i}})^{\tau} g_q^{f_2 v_i} \widehat{R}_{2,i} \right\}_{i=1}^{n} \end{array} \right)$$

- For the d^{th} token query, the simulator will implicitly set the exponents $\vec{r} = \vec{\gamma}$, where $\vec{\gamma}$ is adopted from the ℓ-DLinear instance. More specifically, the simulator computes the following token:

$$\mathsf{TK} = \left(\begin{array}{l} K_0 = R_0 \cdot \prod_{i=1}^{n} (g_p^{\gamma_{1,i}})^{-\omega_{1,i}} (g_p^{\gamma_{2,i}})^{-\omega_{2,i}}, \quad K_\emptyset = R_\emptyset \cdot \prod_{i=1}^{n} (g_p^{\gamma_{1,i}})^{-\kappa_{1,i}} (g_p^{\gamma_{2,i}})^{-\kappa_{2,i}} \\ \left\{ K_{1,i} = g_p^{\gamma_{1,i}} g_q^{f_1 v_i} \widehat{R}_{1,i}, \quad K_{2,i} = g_p^{\gamma_{2,i}} g_q^{f_2 v_i} \widehat{R}_{2,i} \right\}_{i=1}^{n} \end{array} \right)$$

- For the remaining token queries $d+1, \ldots, k$, the simulator generates tokens by directly calling the $GenToken$ algorithm. In this case, the exponents $\vec{r}$ is chosen as a random vector in $\mathbb{F}_p^{2n}$.

- **Guess.** If the adversary guesses that it is playing Game_d, the simulator guesses that $\vec{\gamma} \xleftarrow{R} \mathsf{closure}(\vec{y}, \vec{z})$. Otherwise, if the adversary guesses that it is playing Game_{d-1}, the simulator guesses that $\vec{\gamma} \xleftarrow{R} \mathbb{F}_p^{2n}$.

Clearly, if the adversary has ϵ advantage in distinguishing Game_d and Game_{d-1} ($1 \leq d \leq k$), the simulator also has ϵ advantage in the ℓ-DLinear experiment. ■

Symmetry of token and ciphertext in SCHEMESYM

So far, it may not be completely obvious why the tokens and ciphertexts are symmetrically formed in SCHEMESYM. To show why this is true, we give a different description of SCHEMESYM, and call the resulting scheme SCHEMESYMII. SCHEMESYMII is in fact the same scheme as SCHEMESYM, although the description seems different on the surface. It will be clear from the description of SCHEMESYMII that tokens and ciphertexts symmetrically formed. We then explain why SCHEMESYM and SCHEMESYMII are in fact the same scheme. Basically, tokens and ciphertexts in SCHEMESYM are identically distributed as tokens and ciphertexts in SCHEMESYMII (except with negligible probability).

Before we formally define SCHEMESYMII, we first explain the intuition. In SCHEMESYM, both the ciphertext and token have $2n+2$ terms. Clearly, in SCHEMESYMII, tokens and ciphertexts are symmetric in the $\mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_{\hat{r}}$ subgroups. In particular, the $\mathbb{G}_q$ subgroup has the same form in both the ciphertext and the token, and the $\mathbb{G}_r$ and $\mathbb{G}_{\hat{r}}$ subgroups "mirror" each other.

However, it may not entirely obvious that the $\mathbb{G}_p$ subgroup is symmetric as well; and this is what we are about to show. Let us now focus on the elements in the $\mathbb{G}_p$ subgroup in the ciphertext and token. We represent elements in the $\mathbb{G}_p$ subgroup in the canonical form g_p^x, where g_p is a generator of $\mathbb{G}_p$, and $x \in \mathbb{Z}_p$. In both the ciphertext and the token, the exponents in the $\mathbb{G}_p$ subgroup (base g_p) form a vector in $\mathbb{F}_p^{2n+2}$. We now show that these exponents have the following distribution (except with negligible probability).

- Pick two random 2-dimensional subspaces $S_1, S_2 \subset \mathbb{F}_p^{2n+2}$ that are orthogonal to each other, that is, $S_1 \perp S_2$. The fact that $S_1 \perp S_2$ ensures that the $\mathbb{G}_p$ subgroup cancels out in the $Query$ algorithm.
- For every ciphertext generated, pick a random vector $\vec{\mu} \xleftarrow{R} S_1$ to be the exponents in the $\mathbb{G}_p$ subgroup (base g_p).
- For every token generated, pick a random vector in $\vec{\nu} \xleftarrow{R} S_2$ to be the exponents in the $\mathbb{G}_p$ subgroup (base g_p).

Definition 5.7.19 (SCHEMESYMII) *We define the following encryption scheme henceforth referred to as* SCHEMESYMII. *From the description of* SCHEMESYMII*, it is clear that tokens and ciphertexts are symmetrically formed.*

$Setup(1^\lambda)$: The setup algorithm first chooses random large primes $p, q, r, \widehat{r}$, and creates a bilinear group of composite order $N = pqr\widehat{r}$. Next, it picks generators $g_p,\ g_q,\ g_r,\ \widehat{g}_r$ from subgroups $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_{\widehat{r}}$ respectively. The setup algorithm also needs to pick two orthogonal subspaces from $\mathbb{F}_p^{2n+2}$. To do so, the setup algorithm picks the following random exponents from $\mathbb{Z}_p$:

$$\begin{aligned}
&\vec{\mu}_1 = \left(c_0,\ c_\emptyset,\ \ \{c_{1,i},\ c_{2,i}\}_{i=1}^n\right), \quad \vec{\mu}_2 = \left(d_0,\ d_\emptyset,\ \ \{d_{1,i},\ d_{2,i}\}_{i=1}^n\right)\\
&\vec{\nu}_1 = \left(y_0,\ y_\emptyset,\ \ \{y_{1,i},\ y_{2,i}\}_{i=1}^n\right), \quad \vec{\nu}_2 = \left(z_0,\ z_\emptyset,\ \ \{z_{1,i},\ z_{2,i}\}_{i=1}^n\right)\\
&\text{s.t.} \qquad \forall (i,j) \in [2] \times [2], \quad \langle \vec{\mu}_i, \vec{\nu}_j \rangle = 0
\end{aligned}$$

For example,

$$\langle \vec{\mu}_1, \vec{\nu}_1 \rangle := c_0 y_0 + c_\emptyset y_\emptyset + \sum_{i=1}^{n} \left(c_{1,i} y_{1,i} + c_{2,i} y_{2,i}\right)$$

All of the above parameters are kept as the secret key.

Remark 5.7.3 *Intuitively, by picking $\vec{\mu}_1, \vec{\mu}_2$ and $\vec{\nu}_1, \vec{\nu}_2$, we are effectively picking two random 2-dimensional subspaces in $\mathbb{F}_p^{2n+2}$ that are orthogonal to each other:*

$$\mathsf{closure}(\vec{\mu}_1, \vec{\mu}_2) \perp \mathsf{closure}(\vec{\nu}_1, \vec{\nu}_2)$$

In the unlikely event that $\vec{\mu}_1$ and $\vec{\mu}_2$ (or $\vec{\nu}_1$ and $\vec{\nu}_2$) are linearly dependent, the dimension of $\mathsf{closure}(\vec{\mu}_1, \vec{\mu}_2)$ *(or* $\mathsf{closure}(\vec{\nu}_1, \vec{\nu}_2)$*) may be smaller than 2. However, this happens only with negligible probability.*

$Encrypt(\mathsf{MSK}, \vec{x})$: Let $\vec{x} = (x_1, x_2, \ldots, x_n) \in (\mathbb{Z}_N)^n$. The encryption algorithm first picks random exponents s, t, α, β from $\mathbb{Z}_p$. Then, it chooses random hiding factors $\widehat{R}_0, \widehat{R}_\emptyset$ from the subgroup $\mathbb{G}_{\widehat{r}}$; and random $\{R_{1,i}, R_{2,i}\}_{i=1}^n$ from $\mathbb{G}_r$. The encryption algorithm computes the following ciphertext:

$$\mathsf{CT} = \left(\begin{array}{l} C_0 = \widehat{R}_0 \cdot g_p^{sc_0 + td_0}, \quad C_\emptyset = \widehat{R}_\emptyset \cdot g_p^{sc_\emptyset + td_\emptyset} \\ \left\{ C_{1,i} = g_p^{sc_{1,i} + td_{1,i}} g_q^{\alpha x_i} R_{1,i},\ \ C_{2,i} = g_p^{sc_{2,i} + td_{2,i}} g_q^{\beta x_i} R_{2,i} \right\}_{i=1}^n \end{array} \right)$$

Remark 5.7.4 *In the above ciphertext, the exponents in the $\mathbb{G}_p$ subgroup form the following vector:*

$$\vec{\mu} = \left(sc_0 + td_0,\ sc_\emptyset + td_\emptyset,\ \ \{sc_{1,i} + td_{1,i},\ sc_{2,i} + td_{2,i}\}_{i=1}^n\right) = s\vec{\mu}_1 + t\vec{\mu}_2$$

It is not hard to see that $\vec{\mu}$ is chosen as a random vector in the 2-dimensional subspace defined by $\mathsf{closure}(\vec{\mu}_1, \vec{\mu}_2)$.

$GenToken(\mathsf{MSK}, \vec{v})$: Let $\vec{v} = (v_1, v_2, \ldots, v_n) \in (\mathbb{Z}_N)^n$. The $GenToken$ algorithm behaves symmetrically to the $Encrypt$ algorithm. It first picks random exponents ρ, τ, f_1, f_2 from $\mathbb{Z}_p$. Then, it chooses random hiding factors $R_0, R_\emptyset$ from the subgroup $\mathbb{G}_r$; and random $\{\widehat{R}_{1,i}, \widehat{R}_{2,i}\}_{i=1}^n$ from $\mathbb{G}_{\widehat{r}}$. The token is formed as below:

$$\mathsf{TK} = \left(\begin{array}{l} K_0 = R_0 \cdot g_p^{\rho y_0 + \tau z_0}, \quad K_\emptyset = R_\emptyset \cdot g_p^{\rho y_\emptyset + \tau z_\emptyset} \\ \left\{ K_{1,i} = g_p^{\rho y_{1,i} + \tau z_{1,i}} g_q^{f_1 v_i} \widehat{R}_{1,i},\ \ K_{2,i} = g_p^{\rho y_{2,i} + \tau z_{2,i}} g_q^{f_2 v_i} \widehat{R}_{2,i} \right\}_{i=1}^n \end{array} \right)$$

Remark 5.7.5 *In the above token, the exponents in the $\mathbb{G}_p$ subgroup form the following vector:*

$$\vec{\nu} = \left(\rho y_0 + \tau z_0,\ \rho y_\emptyset + \tau z_\emptyset,\ \ \{\rho y_{1,i} + \tau z_{1,i},\ \rho y_{2,i} + \tau z_{2,i}\}_{i=1}^n\right) = \rho\vec{\nu}_1 + \tau\vec{\nu}_2$$

It is not hard to see that $\vec{\nu}$ is chosen as a random vector in the 2-dimensional subspace defined by $\mathsf{closure}(\vec{\nu}_1, \vec{\nu}_2)$.

$Query(\mathsf{TK}_{\vec{v}}, \mathsf{CT}_{\vec{x}})$: Same as the $Query$ algorithm of SCHEMEREAL. Note that as the two subspaces $\mathsf{closure}(\vec{\mu}_1, \vec{\mu}_2)$ and $\mathsf{closure}(\vec{\nu}_1, \vec{\nu}_2)$ are orthogonal to each other, $\langle \vec{\mu}, \vec{\nu} \rangle = 0$. Hence, in the $Query$ algorithm, elements in the $\mathbb{G}_p$ subgroup cancel out, resulting in $1 \in \mathbb{G}_{T,p}$.

Lemma 5.7.20 (Equivalence of SCHEMESYM and SCHEMESYMII) *Tokens and ciphertexts computed in* SCHEMESYMII *are identically distributed as in* SCHEMESYM *(except with negligible probability).*

Proof: Let us now focus on SCHEMESYM. We first show that in the ciphertext, exponents in the $\mathbb{G}_p$ subgroup are chosen as a random vector in a pre-determined 2-dimensional subspace (also chosen at random) in $\mathbb{F}_p^{2n+2}$.

For $1 \leq i \leq n$, let $\omega_{1,i}, \omega_{2,i}$ denote the discrete log of $h_{1,i}, h_{2,i}$ (base g_p); let $\kappa_{1,i}, \kappa_{2,i}$ denote the discrete log of $\bar{h}_{1,i}, \bar{h}_{2,i}$ (base g_p). $\{\omega_{1,i}, \omega_{2,i}\}_{i=1}^n$ and $\{\kappa_{1,i}, \kappa_{2,i}\}_{i=1}^n$ are chosen independently at random from $\mathbb{Z}_p$ in the $Setup$ algorithm.

In the $Encrypt$ algorithm of SCHEMESYM, we pick two random numbers $s, t \xleftarrow{R} \mathbb{Z}_p$, and in the ciphertext, the exponents in the $\mathbb{G}_p$ subgroup (base g_p) have the following form:

$$\vec{\mu} := \left(s,\ t,\ \ \{s\omega_{1,i} + t\kappa_{1,i},\ s\omega_{2,i} + t\kappa_{2,i}\}_{i=1}^n\right) \tag{5.11}$$

Define the following two vectors:

$$\begin{array}{l} \vec{\mu}_1 := (1, 0,\ \{\omega_{1,i}, \omega_{2,i}\}_{i=1}^n) \in \mathbb{F}_p^{2n+2} \\ \vec{\mu}_2 := (0, 1,\ \{\kappa_{1,i}, \kappa_{2,i}\}_{i=1}^n) \in \mathbb{F}_p^{2n+2} \end{array} \tag{5.12}$$

Equation (5.11) can be expressed in the following form:

$$\vec{\mu} = s\vec{\mu}_1 + t\vec{\mu}_2$$

■

Therefore, an equivalent way to think of SCHEMESYM is as follows. In the $Setup$ algorithm, we pick two vectors $\vec{\mu}_1$ and $\vec{\mu}_2$ as in Equation (5.12). It is not hard to see that $\mathsf{closure}(\vec{\mu}_1, \vec{\mu}_2)$ defines a random 2-dimensional subspace in $\mathbb{F}_p^{2n+2}$ (except with negligible probability). Later, when computing ciphertexts, we always pick the exponents in the $\mathbb{G}_p$ subgroup as a random vector in $\mathsf{closure}(\vec{\mu}_1, \vec{\mu}_2)$.

We now examine the tokens in SCHEMESYM. It remains to show that in the tokens, exponents in the $\mathbb{G}_p$ subgroup are chosen as random vectors from a random 2-dimensional subspace orthogonal to $\mathsf{closure}(\vec{\mu}_1, \vec{\mu}_2)$. It is not hard to see that in the tokens of SCHEMESYM, the exponents of the $\mathbb{G}_p$ subgroup are picked from a subspace orthogonal to $\mathsf{closure}(\vec{\mu}_1, \vec{\mu}_2)$, since in the $Query$ algorithm, the $\mathbb{G}_p$ subgroup always cancels out, resulting in $1 \in \mathbb{G}_p$. Now, we just need to show that the exponents in the token form a 2-dimensional subspace (as opposed to 1 dimension or other number of dimensions.) To understand why this is the case, we now present alternative way to understand the formation of tokens in SCHEMESYM. In the $Setup$ phase, pick the vectors $\vec{y} = (y_0, y_\emptyset, \{y_{1,i}, y_{2,i}\}_{i=1}^n)$ and $\vec{z} = (z_0, z_\emptyset, \{z_{1,i}, z_{2,i}\}_{i=1}^n)$ as below:

1. Pick $2n$ out of the $2n+2$ coordinates at random, that is, pick $\{y_{1,i}, y_{2,i}\}_{i=1}^n$ at random from $\mathbb{Z}_p$.
2. Given the constraints that $\langle \vec{y}, \vec{\mu}_1 \rangle = 0$, and $\langle \vec{y}, \vec{\mu}_2 \rangle = 0$, the first two coordinates $y_0, y_\emptyset$ can be solved through a system of linear equations. We have two linear equations with two indeterminants. The coefficients of the linear equations are linearly independent except with negligible probability. This means that except with negligible probability, $y_0, y_\emptyset$ can be uniquely solved.
3. Pick $\vec{z}$ in exactly the same way as we did for $\vec{y}$.

It is not hard to see that by picking the vectors $\vec{y} = (y_0, y_\emptyset, \{y_{1,i}, y_{2,i}\}_{i=1}^n)$ and $\vec{z} = (z_0, z_\emptyset, \{z_{1,i}, z_{2,i}\}_{i=1}^n)$ in the manner specified above, we are equivalently picking a random subspace that is orthogonal to the subspace $\mathsf{closure}(\vec{\mu}_1, \vec{\mu}_2)$.

Later, when computing tokens, the $GenToken$ algorithm picks the exponents in the $\mathbb{G}_p$ subgroup as a random vector from $\mathsf{closure}(\vec{y}, \vec{z})$.

5.8 Proof of Proposition 5.3.2

Proof of Proposition 5.3.2: Our proof is inspired by the hybrid argument used by Katz et al. [28]. We are given SCHEME_{2n} which is SCI-secure, and our goal is to construct a fully-secure construction SCHEME_n. We give an explicit construction of SCHEME_n below. Let $\vec{x} = (x_1, x_2, \ldots, x_n) \in \Sigma^n$, $\vec{x}' = (x'_1, x'_2, \ldots, x'_n) \in \Sigma^n$ denote two vectors of length n. Define

$$\vec{x} || \vec{x}' := (x_1, x_2, \ldots, x_n, x'_1, x'_2, \ldots, x'_n)$$

to be a vector of length $2n$ obtained by concatenating $\vec{x}$ and $\vec{x}'$. In particular, define

$$\vec{x}||\vec{x} := (x_1, x_2, \ldots, x_n, x_1, x_2, \ldots, x_n)$$

to be a vector of length $2n$ obtained by repeating $\vec{x}$ twice. Informally, our construction of SCHEME_n works as follows. To encrypt a vector $\vec{x}$ of length n in SCHEME_n, we encrypt the vector $(\vec{x}, \vec{x})$ of length $2n$ using SCHEME_{2n}. Similarly, to construct a token for the vector $\vec{v}$ of length n, we use SCHEME_{2n} to construct a token for the vector $(\vec{v}, \vec{v})$ of length $2n$.

$\text{SCHEME}_n.Setup(1^\lambda)$: Call $\text{SCHEME}_{2n}.Setup(1^\lambda)$, and output exactly the same secret key MSK.

$\text{SCHEME}_n.Encrypt(\mathsf{MSK}, \vec{x})$: Call $\text{SCHEME}_{2n}.Encrypt(\mathsf{MSK}, \vec{x}||\vec{x})$ and output the resulting ciphertext.

$\text{SCHEME}_n.GenToken(\mathsf{MSK}, \vec{v})$: Call $\text{SCHEME}_{2n}.GenToken(\mathsf{MSK}, \vec{v}||\vec{v})$ and output the resulting token.

$\text{SCHEME}_n.Query(\mathsf{TK}, \mathsf{CT})$: Call $\text{SCHEME}_{2n}.Query(\mathsf{TK}, \mathsf{CT})$ and output the same outcome.

Note that the above construction is valid due to the following fact.

Fact 5.8.1 *Let $N = pqr\widehat{r}$, where p, q, r and $\widehat{r}$ are distinct large (odd) primes. Let $\vec{x}, \vec{v} \in \mathbb{Z}_N^n$. Then*

$$\langle \vec{x}, \vec{v} \rangle = 0 \quad \textit{iff} \quad \langle \vec{x}||\vec{x},\ \vec{v}||\vec{v} \rangle = 0$$

It remains to show that the above SCHEME_n is fully-secure. To do so, let us first recall the the security game (of full security). An adversary makes a series of queries to a challenger. Each query can be a ciphertext query or a token query. In a ciphertext query, the adversary specifies two vectors $\vec{x}, \vec{y}$ to the challenger, and gets back an encryption of one of these two plaintexts. In a token query, the adversary specifies two vectors $\vec{v}, \vec{w}$ to the challenger, and gets back a token for one of these vectors. Suppose that the adversary makes c ciphertext queries, denoted $(\vec{x}_1, \vec{y}_1)$, $(\vec{x}_2, \vec{y}_2)$, $\ldots$, $(\vec{x}_c, \vec{y}_c)$, and t token queries denoted $(\vec{v}_1, \vec{w}_1)$, $(\vec{v}_2, \vec{w}_2)$, $\ldots$, $(\vec{v}_t, \vec{w}_t)$ respectively. Let $X := (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_c)$ and $Y := (\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_c)$ denote the ciphertext queries made by the adversary. Let $V := (\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_c)$ and $W := (\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_c)$ denote the token queries made by the adversary. Recall that X, Y, V, W must satisfy the "indistinguishability under access pattern" condition:

$$\text{ACCESSPATTERN}(X, V) = \text{ACCESSPATTERN}(Y, W)$$

The challenger has a secret random bit b, and depending on its value, the challenger either constructs ciphertexts/tokens for X, V (referred to as World 0), or constructs ciphertexts/tokens for Y, W (referred to as World 1). Our task is to show that the adversary cannot distinguish between World 0 and World 1. To this end, we construct the following series of hybrid games.

World 0 : The challenger calls SCHEME_{2n} and computes ciphertexts/tokens for:

$$\begin{bmatrix} \vec{x}_1||\vec{x}_1, & \vec{x}_2||\vec{x}_2, & \ldots, & \vec{x}_c||\vec{x}_c \\ \vec{v}_1||\vec{v}_1, & \vec{v}_2||\vec{v}_2, & \ldots, & \vec{v}_t||\vec{v}_t \end{bmatrix}$$

World A : The challenger calls SCHEME_{2n} and computes ciphertexts/tokens for:

$$\begin{bmatrix} \vec{x}_1||\vec{0}, & \vec{x}_2||\vec{0}, & \ldots, & \vec{x}_c||\vec{0} \\ \vec{v}_1||\vec{v}_1, & \vec{v}_2||\vec{v}_2, & \ldots, & \vec{v}_t||\vec{v}_t \end{bmatrix}$$

World B : The challenger calls SCHEME_{2n} and computes ciphertexts/tokens for:

$$\begin{bmatrix} \vec{x}_1||\vec{0}, & \vec{x}_2||\vec{0}, & \ldots, & \vec{x}_c||\vec{0} \\ \vec{v}_1||\vec{w}_1, & \vec{v}_2||\vec{w}_2, & \ldots, & \vec{v}_t||\vec{w}_t \end{bmatrix}$$

World M : The challenger picks random $\alpha \xleftarrow{R} \mathbb{Z}_N$, calls SCHEME_{2n} and computes ciphertexts/tokens for the following vectors:

$$\begin{bmatrix} \vec{x}_1||\alpha\vec{y}_1, & \vec{x}_2||\alpha\vec{y}_2, & \ldots, & \vec{x}_c||\alpha\vec{y}_t \\ \vec{v}_1||\vec{w}_1, & \vec{v}_2||\vec{w}_2, & \ldots, & \vec{v}_t||\vec{w}_t \end{bmatrix}$$

Remark 5.8.1 *Notice that in the above hybrid sequence, the access pattern remains the same between all worlds except with negligible probability.*

Claim 5.8.2 *Assume that* SCHEME_{2n} *is SCI-secure, then no polynomial-time adversary has more than negligible advantage in distinguishing between adjacent games.*

Proof: By hybrid argument. ■

Similarly, we can have a sequence of hybrid games connecting World 0 and World M. Due to the hybrid argument, we conclude that no polynomial adversary has more than negligible advantage in distinguishing World 0 and World 1. ■

5.9 Comparison with Previous Security Definitions

Only two prior works have considered query privacy in SK-PE: the work by Song et al. [39], and the work by Curtmola et al. [19]. In addition, both of these works consider simple keyword-based queries. Song et al. were the first ones to propose a searchable encryption scheme, and they did not present a formal security definition for query privacy. Curtmola et al. presented a formal security definition to intuitively capture the notion that both the plaintext entries and the queries should be hidden from the storage server. Their security definition is not satisfactory due to the following reasons:

- The security definition by Curtmola et al. reveals the "search pattern", that is, if a user issues two queries for the same keyword, the storage server learns the fact that these two queries are equal. In our security definition, the query has the same (selective) semantic security as the plaintext. In particular, let A and A' denote two queries with the same access patter, then the storage server is unable to decide whether a user has made the same query A twice, or whether the user queried for A followed by A' instead.
- In Curtmola's security definition, the adversary first commits to two sets of documents (denoted $\mathcal{D}_0, \mathcal{D}_1$ in their paper [19]). Using our terminology, this means that the adversary has to commit to all the ciphertext queries it intends to make. Instead, we give the adversary more power in our full security definition. The adversary should be fully adaptive: it can decide what ciphertext queries and token queries to make depending on previous interactions with the challenger.

- Although we did not prove the security of our construction under the full security model, As Observation 5.9.1 points out, even the relaxed security model actually used in our proofs is stronger than Curtmola's definition.

Proposition 5.9.1 *Given a selectively SCI-secure SK-PE construction on inner-product queries for vectors of length $2n$, it is possible to construct an SK-PE scheme on inner-product queries for vectors of length n, satisfying the security definition by Curtmola et al. (Definition 3.8 in their paper [19]).*

Proof: The above proposition can be proved in a similar manner as Proposition 5.3.2. ■

In fact, in the above proof, when we use the scheme for vectors of length $2n$ to construct a scheme for vectors of length n, the resulting scheme (for vectors of length n) has stronger security than Curtmola's definition, as Curtmola's definition reveals the search pattern in addition. To reiterate, revealing the search pattern means that the storage server can tell if two queries submitted by the user are the same or not. Our security definition does not reveal the search pattern.

5.10 Review of the KSW Construction

To aid the understanding of our construction, we review the KSW construction [28] for inner-product queries in the public-key setting.

$Setup(1^\lambda)$: The setup algorithm first chooses random large primes p, q, r, and creates a bilinear group of composite order $N = pqr$. Next it picks generators $g_p,\ g_q,\ g_r$ from subgroups $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r$ respectively. It also picks $h_{1,i}, h_{2,i}$ from $\mathbb{G}_p$, $R_{1,i}, R_{2,i}$ from $\mathbb{G}_r$ for all $1 \leq i \leq n$, and random R_0 from $\mathbb{G}_r$.

The public key is composed as below:

$$\mathsf{PK} = (g_p,\ g_r,\ Q = g_q \cdot R_0,\ \ \{H_{1,i} = h_{1,i}R_{1,i},\ H_{2,i} = h_{2,i}R_{2,i}\}_{i=1}^n)$$

The secret key is set to the following:

$$\mathsf{Pvk} = \big(p, q, r,\ g_q,\quad \{h_{1,i}, h_{2,i}\}_{i=1}^n\big)$$

$Encrypt(\mathsf{PK}, \vec{x})$: Let $\vec{x} = (x_1, x_2, \ldots, x_n) \in (\mathbb{Z}_N)^n$. The encryption algorithm first picks random exponents s, α, β from $\mathbb{Z}_N$, and it chooses random $\{R_{3,i}, R_{4,i}\}_{i=1}^n$ from $\mathbb{G}_r$.

Next, the encryption algorithm computes the following ciphertext:

$$\mathsf{CT} = \left(\begin{array}{l} C_0 = g_p^s, \\ \big\{C_{1,i} = H_{1,i}^s Q^{\alpha x_i} R_{3,i},\ \ C_{2,i} = H_{2,i}^s Q^{\beta x_i} R_{4,i}\big\}_{i=1}^n \end{array} \right)$$

$GenToken(\mathsf{MSK}, \vec{v})$: Let $\vec{v} = (v_1, v_2, \ldots, v_n) \in (\mathbb{Z}_N)^n$. The $GenToken$ algorithm picks random exponents $f_1, f_2, \{r_{1,i}, r_{2,i}\}_{i=1}^n$ from $\mathbb{Z}_N$. Then, it chooses a random hiding factor R_5 from the subgroup $\mathbb{G}_r$, and random Q_6 from $\mathbb{G}_q$.

Next, the $GenToken$ algorithm computes the following token:

$$\mathsf{TK} = \left(\begin{array}{l} K = R_5 Q_6 \cdot \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \\ \left\{ K_{1,i} = g_p^{r_{1,i}} g_q^{f_1 v_i},\ K_{2,i} = g_p^{r_{2,i}} g_q^{f_2 v_i} \right\}_{i=1}^{n} \end{array} \right)$$

$Query(\mathsf{TK}_{\vec{v}}, \mathsf{CT}_{\vec{x}})$: The $Query$ algorithm computes

$$\mathsf{e}(C_0, K_0) \cdot \prod_{i=1}^{n} \mathsf{e}(C_{1,i}, K_{1,i}) \mathsf{e}(C_{2,i}, K_{2,i}) \stackrel{?}{=} 1 \tag{5.13}$$

and outputs 0 iff the above is equal to 1, indicating that $\langle \vec{x}, \vec{v} \rangle = 0 \mod N$. (The case that $\langle \vec{x}, \vec{v} \rangle = 0 \mod q$, but $\langle \vec{x}, \vec{v} \rangle \neq 0 \mod N$ happens with negligible probability as explained in Section 5.6.)

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Predicate encryption is a new encryption paradigm enabling fine-grained access control to the encrypted data. In predicate encryption, the secret key owner can compute a capability which allows one to evaluate the outcome of a predicate on the encrypted data.

An important research challenge in predicate encryption is how to support more expressive query predicates and richer operations. In this thesis, we made the following contributions to the area of predicate encryption.

- We propose a predicate encryption scheme supporting *multi-dimensional range queries* (Chapter 3). This construction is secure in the match-revealing model. Multi-dimensional range queries is particularly important in practice, especially in database applications, as SQL queries are by nature multi-dimensional range queries.
- We study how to *delegate* capabilities in predicate encryption, and propose a construction that supports delegation on conjunctive queries (Chapter 4).
- We consider the problem of *query privacy* in predicate encryption. In many practical applications, it would be desirable to hide the queries encoded in the capabilities, in addition to hiding the plaintext data. We show that query privacy is inherently not possible in the public-key setting, due to the fact that anyone can encrypt with a public key. However, we demonstrate that query privacy is indeed possible in the secret-key setting. Specifically, we provide a secret-key predicate encryption scheme that protects the privacy of both the query predicates and the plaintext data. Our construction supports inner-product queries (Chapter 5).

6.2 Future Work

The following are important questions that remain to be answered in predicate encryption:

- Almost all of the known constructions are proven secure in the selective security model, that is, the adversary commits to a challenge identity at the beginning of the game. In

the setting of bilinear groups, some progress has been made recently at proving adaptive security [22, 23]. In particular, Gentry's construction [22] implies a predicate encryption system on equality-test queries. An important open question is to how to construct more expressive predicate encryption schemes and prove security under the adaptive notion of security.

- Another topic worth investigating is how to build expressive predicate encryption systems using other mathematical primitives and assumptions. For example, Boneh et al. built anonymous identity-based encryption based on the quadratic residuosity problem modulo an RSA composite. This implies a predicate encryption system supporting equality-test queries. It is an open research problem how to build more expressive predicate encryption systems without pairings.
- The most expressive predicate encryption system known to this day supports inner-product queries. A big question is how to build systems that are even more expressive. Based on experience, we know that supporting disjunctions might be hard in pairing-based predicate encryption systems. The inner-product scheme can support bounded-size disjunctive queries by converting them to polynomial evaluation queries. However, such conversions incur a large expansion factor in the cost, making it expensive to support large disjunctive queries.

Bibliography

[1] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In *CRYPTO*, 2005. 1.2, 1.4, 1.5, 3.2.2, 5.1

[2] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, 2007. 1.2

[3] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In *EUROCRYPT*, 2004. 2.1.2, 3.1.3, 4.1.2, 5.1, 5.3.3

[4] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005. 4.1.2, 4.4

[5] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004. 3.3.1, 5.4.2

[6] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004. 1.2, 1.4, 1.5, 5.1

[7] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Proceedings of Crypto 2001*, volume 2139 of *LNCS*, pages 213–29. Springer-Verlag, 2001. 1.2

[8] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003. 3.3.1

[9] Dan Boneh, Craig Gentry, and Michael Hamburg. Space-efficient identity based encryption without pairings. In *Proceedings of FOCS*, 2007. 1.2, 1.4, 1.5, 5.1

[10] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, *Proceedings of Theory of Cryptography Conference 2005*, volume 3378 of *LNCS*, pages 325–342. Springer, 2005. 5.4.1

[11] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, pages 573–592, 2006. 4.6.5

[12] Dan Boneh and Brent Waters. A fully collusion resistant broadcast trace and revoke system with public traceability. In *ACM Conference on Computer and Communication Security (CCS)*, 2006. 1.2, 1.5, 1.2, 2.1, 2.1.2, 3.1.1, 4.1.1, 4.1.2, 4.1.3, 4.1, 4.2, 4.2.2, 4.3, 4.4, 4.4.1, 4.5, 4.6.5, 5.1, 5.1, 5.3, 5.3.3, 5.4.2

[13] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *CRYPTO*, 2006. 1.2, 1.5, 2.1.2, 3.2.2, 3.3.4, 3.7.2, 4.1.2, 4.1.2, 4.8, 4.8.2, 5.1, 5.1, 5.3.3

[14] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003. 2.1.2, 3.1.3, 4.1.2, 5.1, 5.3.3

[15] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT*, pages 207–222, 2004. 2.1.2, 3.1.3, 4.1.2, 5.1, 5.3.3

[16] Melissa Chase. Multi-authority attribute based encryption. In *TCC*, pages 515–534, 2007. 1.2

[17] The Elliptic Semiconductor CLP-17 high performance elliptic curve cryptography point multiplier core: Product brief. `http://www.ellipticsemi.com/pdf/CLP-17_60102.pdf`. 3.3.5

[18] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, pages 360–363, London, UK, 2001. Springer-Verlag. 1.2

[19] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, 2006. 5.1, 5.3.4, 5.9, 5.9.1

[20] Symantec deepsight threat management system technology brief. `https://tms.symantec.com`. 1.3

[21] The dshield project. `http://www.dshield.org`. 1.3

[22] Craig Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, 2006. 1.2, 1.5, 6.2

[23] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems. Technical report, 2008. E-print archives, `http://eprint.iacr.org/2008/268.pdf`. 6.2

[24] Oded Goldreich. Secure multi-party computation. Volume 2, Foundations of Cryptography, 1998. 1.3

[25] Philippe Golle, Jessica Staddon, and Brent R. Waters. Secure conjunctive keyword search over encrypted data. In *ACNS*, pages 31–45, 2004. 1.5, 5.1

[26] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security (CCS)*, 2006. 1.2

[27] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In *ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394, London, UK, 2000. Springer-Verlag. 3.3.1

[28] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Eurocrypt '08, to appear*, 2008. 1.2, 1.5, 4.1.1, 5.1, 5.3, 5.4.2, 5.4.2, 5.5.1, 5.5.2, 5.5.1, 5.7.10, 5.7.11, 5.7.14, 5.8, 5.10

[29] Ben Lynn. The Pairing-Based Cryptography (PBC) library. http://crypto.stanford.edu/pbc. 3.3.5

[30] The mynetwatchman project. http://www.mynetwatchman.com. 1.3

[31] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure attribute-based systems. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, 2006. 1.2

[32] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005. 1.2, 1.3

[33] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of Crypto*, 1984. 1.2

[34] Emily Shen, Elaine Shi, and Brent Waters. Query-hiding secret-key predicate encryption with inner-product queries. manuscript. 1.2, 1.5, 1.5

[35] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimension range query over encrypted data. In *IEEE Symposium on Security and Privacy*, May 2007. 1.2, 1.5, 1.5, 1.2, 2.1.2, 2.1.3, 3.1, 3.1.1, 4.1.2, 5.1, 5.1, 5.3, 5.3.3

[36] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In *ICALP*, 2008. 1.2, 1.5, 1.5

[37] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In *Proceedings of ICALP*, 2008. Full version can be found online at http://sparrow.ece.cmu.edu/~elaine/docs/delegation.pdf. 4, 5.1, 5.7.3

[38] skrenta. The secret source of google's power. http://blog.topix.com/archives/000016.html 3.3.5

[39] Dawn Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE symposium on Security and Privacy*, 2000. 1.4, 1.5, 5.1, 5.9